The Use Of Deep Learning To Solve Invariance Issues In Object Recognition

Barrow, Erik

Award date:
2017

Awarding institution:
Coventry University

The Use Of Deep Learning To Solve Invariance Issues In Object Recognition

Erik Jonathan Barrow

Faculty of Engineering, Environment, and Computing

Coventry University

This dissertation is submitted for the degree of

Doctor of Philosophy

School of CEM

September 2017

I dedicate my Thesis to my family who have supported me throughout my studies. Firstly to my Grandmother Jean Barrow who passed away from cancer before I started University, but always encouraged me to do more and to do my best while not giving up on things I find difficult.

Secondly I also dedicate my thesis to my remaining Grandparents Dudley Barrow, Ron Goldfinch, and Margaret Goldfinch, who where always available for a phone call if work was getting tough.

Thirdly and finally, I also dedicate my thesis to my parents, Chris Barrow, and Susan Barrow, as well as my aunt, Mandy Barrow, and my great aunt, Ruth Barrow, who have all supported me in undertaking my undergraduate degree and my postgraduate studies.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification at Coventry University, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains approximately 31535 words, 58 figures, 10 tables, and 32 equations.

Erik Jonathan Barrow
September 2017

Acknowledgements

I would like to thank my Director of Studies, Dr Mark Eastwood for the immense support he has given me with my work and helping guide me through roadblocks and problems that occurred during the project lifespan, as well as helping teach me the fundamentals of neural networks and machine learning that allowed me to undertake this project.

I would also like to acknowledge the rest of my supervisory team, Dr Chrisina Jayne and Dr Vasile Palade, who also provided me with support during my project.

I would also like to thank my fellow postgraduate researchers, Luke Hicks, Ariel Ruiz-Garcia, and Ibrahim Almakky, for their support and helping in keeping me sane throughout my postgraduate project, as well as being there to help bounce around new ideas.

Abstract

Object recognition is a complex problem in the field of computer vision and artificial intelligence with many different solutions including the use of Neural Networks and Deep Learning. This thesis investigates the use of Deep learning for object recognition and improvements to the deep learning methodology using methods such as dropout to reduce overfitting. Improvements to the dropout method are made using selective neuron dropping instead of random neuron dropping used in traditional uniform dropout to improve results up to 1.17% on the CHARS74K dataset using neuron output variance to select neurons for dropping. This thesis also explores the use of Deep Convolutional Neural Networks to create a rotation invariant neural network by using parallel convolutional layers with tied weights with filters presented at different rotations, combining results using a winner takes all pooling method to produce rotation invariance. Results of this method showed great improvement over benchmarks on rotated test data showing a 52.32% increase in accuracy on the MNIST dataset and a 36.44% accuracy increase on the CHARS74K dataset. This work was furthered by attempting to introduce a scaling method into the filter rotation to reduce data loss and blurring by the rotation method, however it was found that results using this method worsened the accuracy of the network compared to benchmarks not using the method, showing a decrease in accuracy of 0.17% on the MNIST dataset and a decrease of 3.68% on the CHARS74K dataset.

Table of Contents

Chapter 1

Introduction

1.1 Introduction to the Thesis

1.1.1 Thesis Statement

This thesis looks at the use of machine learning for object recognition in machine vision, specifically looking at the use of deep learning to create an invariant neural network which will reduce error rates when images containing transformations different to the training set are presented.

Object recognition is a difficult task in computer vision, which gets even more complicated when you consider all the different variations of an object there are, along with an almost infinite amount of ways that object can be presented in an image. A machine learning algorithm attempts to learn patterns in images which are typically presented to it as either a vector of pixels, or as a set of features that have been extracted from the image. Thinking about this from the standpoint of a human being would be very difficult. While our brain can interpret automatically the input of our eyes in a similar way, we would struggle to see the data presented to us in a numerical format. Neural networks aim to try and replicate what we understand of the human brain to allow it to learn and recognise objects in a similar way to humans.

While progress has been made in recognising objects in images, variance in images is still the biggest problem that causes a network to incorrectly classify an object [71]. Not only can there be many types of object that are part of the same category (just think of how many different shaped cups you can think of), but the same object can look very different to a computer just by changing its pose, position, or lighting in an image. Transformations in an image can also then further deepen the problem when they are scaled, rotated, translated, or warped.

Some methods use approaches like brute force to train a network on as many variations as possible, but this is not an ideal solution as there is no way to cover all the possibilities, and requires a extremely large dataset and network to learn.

A method of recognising objects that are rotated in an image is presented for this thesis using convolutional neural networks, that allows a network to be trained on an upright non-rotated dataset, but still recognise objects that are presented after training that have been rotated in the image. New methods for improving the dropout method by selectively dropping neurons in a dropout neural network rather than randomly are also presented.

1.1.2 Motivations

The use of machine learning to recognise objects in images is a very important field of research. To be able to accurately identify objects without the need for human interaction gives a wide range of applications that can help and improve the lives of many, as well as automating tasks that would be too repetitive or are too large to employ a workforce of humans to undertake.

Object recognition can be applied to may aspects of life. Such applications include areas such as security, space exploration, medical diagnosis, and human assistance. It is also a gateway into creating artificially intelligent robotics that are capable of autonomously navigating and negotiating the physical world.

The field of deep learning is often considered a step back towards adaptive artificial intelligence. A machine that can learn and make its own decisions based on data it has gathered, without the need for human supervision. Depending on your definition of an intelligent machine, and the application of a Turing Test, I believe that a machine that can learn and make decisions based on its own experiences is a step back towards a truly intelligent machine.

The motivations for creating an invariant machine learning method are due to the complexity of recognising images that have been transformed in some way from what a machine learning method has been taught. This thesis looks at rotational invariance to recognise objects that may not be upright in an image, or to recognise objects in situations where there is no up or down, such as in space or with a top down view found with areal photography.

1.1.3 Objectives

The objectives of this thesis are as follows,

- Develop an understanding of the Rotation Invariance problem and its effect on Deep Learning.

- Create original methods to help with combating issues associated with rotation invariance.
- Create ways of improving the rotation invariance properties of existing machine learning methods.
- Improve existing Deep Learning methods that could be used with the Rotation Invariance problem.

1.1.4 Applications

There are many possible applications for deep learning, some of which are already being explored and others which have not. One application for computer vision and recognition is security. A computer could go through thousands of images and watch hundreds of security cameras simultaneously looking for suspicious behaviour or prohibited items. An existing application on this can be shown by a company that uses a robot security guard [93], which surveys rooms looking for abnormalities by looking at any differences from previous scans.

Deep learning can also be used with face recognition [103]. This is a good security feature that could be used for logging into machines, opening doors, or even by security companies to detect unauthorised entry. It could also be used with robots in the future, that have a need to identify who they are talking to or allow a user to identify someone. Expanding on security, a face recognition algorithm could allow suspects and criminals to be identified automatically on a vast array of security cameras around the world, and notify human operatives of their location.

Deep learning and object recognition could also be used with heads-up displays, such as Google glass [56]. Possibilities exist to overlay information onto what the user sees using augmented reality, such as what the user is looking at, who they are talking to, and providing information about what they are looking at. There are applications with this for disabled users, such as the autistic. The HUD could recognise the emotion of who they are talking to, to allow the user to react accordingly.

Generically looking at just object recognition, Google currently deploy an app called Google Goggles [57], which allows a users to take a photo of an object, and Google will attempt to recognise what is in the picture. This is useful if you are looking at something new or even don't know the name of an object. A newer version of this developed by Samsung is called Bixby Vision [121], which helps users identify objects, landmarks, and businesses, so that the user may learn more about them.

This can lead onto shopping applications, by simply showing the camera the object that you want to buy, or that you are running out of in the kitchen, you could have it added to an

online shopping list. This is currently implemented in some applications by using an objects bar-code, but could be improved with object recognition by even recognising automatically when you are running out of an item. Amazon are currently working on a store with no checkouts called Amazon Go [4], which recognises a user and identifies objects they add to their cart, and automatically bills users electronically when they leave a store.

A very large field that is attempting to use object recognition with some form of learning is robotics. Creating a self-sustaining robot that does not need human control is very difficult. The use of machine learning techniques can help the robot to learn routes, recognise objects and what they do, and to learn about new objects in its environment. Boston dynamics uses computer vision with bipedal robots and four legged robots to create some truly amazing robotics that can be used in a multitude of real world applications [16].

Another useful application would be the ability to tag data automatically. From recognising who is in a photo, to tagging images with the content of the photo. This is currently being used by Facebook when you upload photos, where it will automatically try to recognise faces, and if possible give recommendations of who is in the picture [44]. This would also be useful for a company such as Google that has a large image base, and wants to accurately tag images to allow users to search for them. Google reverse image search is another current application that uses machine learning to allow users to present Google with an image and ask it to find all the similar images, while also tagging what it thinks the image is.

1.1.5 Contributions to the Field

A number of contributions to the field are made in this thesis either through new methods or new applications of existing ideas. Chapters 3, 4, 5, and 5.8 introduce in detail these contributions and how research was conducted to come to their conclusion.

Four main contributions in this thesis can be identified is follows:

- The application of Deep Dropout Networks to natural images of digits and characters. Chapter 3.

- New methods of Selective Dropout to improve dropout performance. Chapter 4. This contribution improves on the already existent dropout method by turning the dropout process from a random one to a selective one.

- The application and creation of Convolutional Neural Networks with parallel columns of rotated filters, along with an implementation in the Caffe machine learning framework. Chapter 5. This contribution allows for convolutional neural networks to become invariant to rotation transformations even when trained on non-rotated training data.

- An investigation on whether scaling a filter before and after rotation in the above contribution can improve recognition. Chapter 5.8.

1.2 Ethics

With this project there are ethics to consider. This involves the ethics of the possible uses of the outputs of this thesis, as well as the ethics of the use of any data obtained for use with experiments and testing for this thesis.

There has recently been media interest in the use of artificial intelligence in machines, specifically around what effect the invention of fully intelligent machines would have on the human race, with well known scientists such as Stephen Hawking weighing in on the implications of artificial intelligence [20]. There are also implications of automation causing large work forces to be made redundant and being replaced with machines. Like many technologies, there are also military applications to artificial intelligence research, and with computer vision in particular allowing for the identification of targets in missile systems as well as in anti-missile defence systems.

Despite issues such as these the field continues to research in the artificial intelligence arena with the knowledge that like most technologies, there are good and bad applications. Some countries and groups around the world are now starting to research problems that may come from increased artificial intelligence around the globe and look into ways to combat the problems those advances may cause. Some such research is that of universal income, which looks at giving a basic income to the entire population to live on, whether they are working or not. This looks to solve not only the problem of poverty, but also the increasing problem of people being made redundant by machines.

1.2.1 Ethical Approval

This project has been granted ethics approval. A copy of the ethics approval can be found attached in Appendix C.

Chapter 2

Literature Review

2.1 Chapter Introduction

This chapter covers the background knowledge and fundamental methodologies needed to be able to understand the methods introduced in this document. This chapter contains information on Invariance and what it is (Section 2.3.4), Machine Vision (Section 2.3, Boltzmann Machines (Section 2.4.2.1), Neural Networks (Section 2.4), and Related Works (Section 2.5).

2.2 Invariance

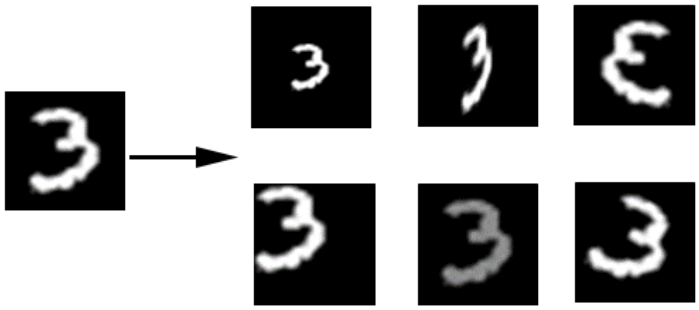
Transformations in images and objects can be considered as one of the main major obstacles to be overcome for object recognition in real world computer vision. Whether trying to find and identify objects in images or video it can be a complex task to deal with all the different permutations or possibilities that the object could be presented in. As humans we invariantly recognise objects in microseconds without much effort or thought to the process [37]. As humans we also have extra well known senses (smell, touch, hearing, and taste) which give us an extra layer of information to help us perceive the world around us, as well as an internal gyroscope that allows us to tell which way up we are in relation to gravity. While some of these senses can be implemented into technology to provide extra information for classification purposes, it can be difficult to get the data into a useful form for use with machine learning methods.

Transformations in the appearance of an object in an image such as rotation, translation, scale, noise, clutter, pose, lighting, and partial viewing [58], makes the task of recognising an image with machine learning a challenging one. Not only can these transformations make

it difficult to recognise an image, but the introduction of the fact that there are many different versions of an object, such as a cup with different pictures on the side, or slightly different shapes, all combine to make a vast number of possible combinations of transformations in training samples. Taking all these combinations into account can create a vary large and computationally complex solution to the object recognition problem. In primates half of the entire cerebral cortex [46] is involved in processing visual information into useful information, suggesting that there is a large level of computational load necessary for such a complex task [58].

The transformation problem in object recognition is often referred to in literature as the "invariance problem" [114][42], and finding a robust and scalable machine learning solution that works for real world images is still considered a large challenge [37].

Fig. 2.1 An example of image variance on the number three



Invariance can be considered as a property of a function. For example, the function f takes an image x as an input and gives an output $f(x)$. If the image x is rotated or scaled with a function g, and $g(x)$ is given as an input to f, then if $f(g(x)) = f(x)$ we say that f is invariant under g. Due to the complexity of this issue a majority of machine learning methods are not invariant under common transformations like rotation, scaling, and translation.

Numerous attempts have been made at dealing with invariance in computer vision. Some of the methods are based on hand-designed features such as SIFT (Scale Invariant Feature Transformation) and HOG (Histogram of Oriented Gradients) [47]. However, these only capture low-level edge information, and finding features related to edge intersections or the different parts that make up an object is more difficult.

Other attempts at dealing with transformations of an object in image recognition have been made by adding a number of typical transformations to the object recognition training process [127]. This method can cause the training period for object classification to exponentially

grow as you add more types of variance to the training process. This is essentially a brute force method which is described in more detail in Section 2.2.2.1.

This section will outline the types of transformations found in computer vision as well as investigate some existing methods which can be used to create an invariant object recognition machine learning tool. Chapter 5 looks at rotation invariance in convolutional neural networks and attempts to improve on the methods presented in this section.

2.2.1 Types of Transformations

There are various types of transformations that can occur to an image that makes the image difficult to recognise. Some of these are a common occurrence while others can be rare, such as image warping due to a camera fault. This section looks at a number of possible transformations that could effect an image, while the thesis focuses on solving issues associated with rotation transformations.

2.2.1.1 Illumination

Illumination on an object in an image causes difficulty for machine learning algorithms. Brighter or darker images can cause a variation in pixel intensities that cause a machine to think that the represented colour or black/white value is different to the original object to be identified. Figure 2.2 shows an example of three different lighting intensities on the same object.

Extra difficulty can be presented when only a portion of an image is affected, creating a large contrast to an object, which may make it seem like multiple objects. An overload of light could also make the object start to blend in with the background of an image, making the process of identifying the object much harder.

In real systems, objects that a learning algorithm is being trained to recognise can be uniformly inputted with a similar lighting value to make the training data uniform, while the test data or real data could be far from this. Brute force could be used to cover a wide variety of possible lighting possibilities, but this would make the training set extremely large as well as start to distort the image representation.

Fig. 2.2 Differences in Illumination

2.2.1.2 Translation

Translation can be a regularly occurring transformation to an object in an image. Translation is where an object appears in an image, whether it is centred or to one side of the image. This can present multiple problems to image recognition systems, primarily because they are usually trained to recognise an object in the centre of an image, not off to the side of an image. Figure 2.3 shows an image that has had a transformation applied to it from the original, which would appear as a different matrix of pixels to a machine learning algorithm.

Brute force could again be applied to this problem by training with a number of translation transformations of an object, or by applying multiple translation transformations to an image to be identified. But like the previous section, this makes the problem very time consuming and unreliable.

Fig. 2.3 Translation Variance (Image from [83])

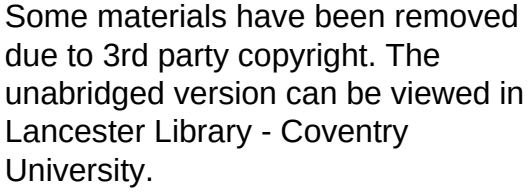
2.2.1.3 Rotation

Small rotations can occur in images depending on the angle of the surface an object is on or the angle of the camera. Camera shake from a user can also cause small rotations of an object in an image. Large rotations can also occur less regularly, perhaps when a camera has been rotated to take the image in portrait or landscape orientation. Another case of large amounts of rotation can occur in situations where there is no natural orientation such as areal imagery and space imagery.

Similar to translation this means that the pixels in the image related to the object will appear in a different location of the image matrix. Small rotations are easier to deal with than large rotations as they are closer to the intended positions, but are still difficult to identify.

Some objects could also look entirely different at a different rotation, and could cause the object to be identified as something different. This can regularly occur for humans when trying to read the number 9 or 6 with no direction indication of up or down. Figure 2.4 shows an example of object rotation.

Fig. 2.4 Rotation Variance (Image from [83])

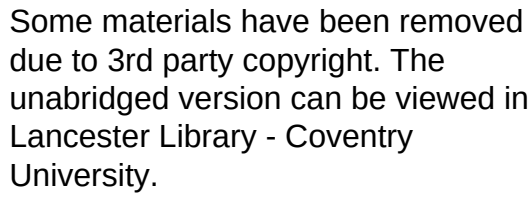


2.2.1.4 Scaling

A change in scale can occur in an image based on how far a camera was from an object, or if the object is a different size to the training set but still the same type of object. Like previous transformations, the scale of an image will make a pixel matrix look entirely different to a machine learning algorithm.

Figure 2.5 shows an example of scaling in an object. Like previous instances brute force could be applied to provide a training set with different possible scales, but would produce a larger and more unreliable training set. Scaled down objects will also provide less information to a machine learning method as a object will take up less pixels for the same amount of data, and as such can blur the data and cause data loss.

Fig. 2.5 Scaling transformations (Image from [83])



2.2.1.5 Perspective Projection

Perspective projection is one of the more difficult transformations in object recognition. There are several instances of perspective projection that can occur to an image of an object, and both are to do with the cameras perspective of an object.

The first possible occurrence is the cameras perspective of the object. This is from what location in space is the camera viewing the object from. An object can look different depending on the perspective it is viewed from in a 3-dimensional space, for example the front and back of a car can look very different, or an image of a cup with handle can look different

if the handle is or is not in view. Figure 2.6 shows an example of changing perspective of an object.

Fig. 2.6 Perspective Projection (Perspective)

Another possible occurrence of perspective projection is distance from an object in a 3-dimensional space. The distance from an object can alter the size of the object in an image, and without the distance data two different objects could be identified as similar, providing a problem similar to that of scaling in Section 2.2.1.4.

An example of this could be an image of a cow in the distance, that looks similar in size to an image of a toy cow, or another animal that would usually be identified as smaller. The difference between a toy and the real thing is not too large a problem if you are not worried about identifying the type of an object, but can cause issues when you do want to be able to differentiate between types of an object. Figure 2.7 shows an example of a cow and toy cow that can look similar in size.

Fig. 2.7 Distance Perspective Projection (Image From [104][100])

2.2.1.6 Deformation

Deformation is another one of the harder variances to recognise, as the object can be warped and distorted from what it normally looks like. There are a number of different types of deformation that can occur to an object in an image [123], each presenting a unique challenge in recognition.

Fig. 2.8 Example of Deformation (Image from [83])

Some materials have been removed due to 3rd party copyright. The unabridged version can be viewed in Lancester Library - Coventry University.

Axis Deformation Axis deformation is a deformation of an image by stretching or shrinking that image along one of its axes. This can happen if either an object has been squashed before a photo, or if the image itself has been stretched or shrunk. This could occur when pre-processing an image or if obtaining images from a website where they have been incorrectly re-sized.

Diagonal Deformation Diagonal deformation is similar to axis deformation, however the image is stretched or shrunk diagonally on the axis. The image starts to warp diagonally and lines that were straight on the x/y axis start to bend.

Thickness Deformation Thickness deformation is where the pixel thickness of the pixels in the object are made bigger, much like the difference between text that is or is not bold. The image has stayed the same size as the training data, but the object has much thicker or thinner lines.

While thicker lines can help with making an object more recognisable and making it stand out, without having done the same to the training set it will only provide variance between the training data and the real world data. Thinner lines make it increasingly difficult to find the outline of an object in the image, and lines that are too thick can cause details to be lost.

2.2.1.7 Summary

While this section has introduced a number of different types of transformations that can be found occurring in images which will cause difficulties for machine learning methods, this thesis focuses on finding solutions to rotation transformations in images. The method presented in Chapter 5 creates a rotation invariant convolutional network, which also has some invariance to translation/shift due to the nature of convolutional neural networks. Future work with the method also explores the possibility for it to be used to create a scale invariant method.

2.2.2 Invariant Representations

There are a number of methods that can be used to try and reduce variance in images to ready them for application to a machine learning algorithm. This can come in several forms such as extracting extra data from the image that is not affected by variance or removing extra data such as colour. The following sections discuss some such methods.

2.2.2.1 Brute Force

As discussed briefly in Section 2.2.1, brute force is one method that is used to produce an invariant machine learning algorithm. Brute force works by taking each image in the training set, and producing another set of images that have been altered to simulate possible possible transformations to that image [73].

While this can force a machine learning algorithm to learn all the representations of an object it is very time consuming. A dataset can also quickly grow to an unmanageable size while trying to simulate all possible inputs and combinations. This will make training very slow, but also has the possibility to deform the machines representation of the object.

Another problem with brute force would be that a larger network may be needed to represent the same data but at different transformations, this in turn adds to training time which can start to grow at an exponential rate. Due to all these issues brute force is seen as a bad solution to the problem and as such this thesis looks at methods which do not create a need to brute force the input data into a network.

2.2.2.2 Feature Extraction

The next current method for recognising objects in images with a large amount of variance is to extract invariant features from the images [97]. Some of these features are invariant to one or more types of variance. Some of the features may not be invariant at all, but can become invariant when combined with other features.

Features extracted from images can be passed as an input for a neural network for training and multiple features are usually combined as typically singular features would have difficulty to make an accurate prediction on their own. The disadvantage of feature extraction is that you have to extract features from new images when classifying them, though this is a similar required step to pre-process images for a network, such as grey-scaling.

2.2.2.3 Histogram of Orientated Gradients

Histogram of Orientated Gradients is a feature descriptor that counts the number of occurrences of a particular gradient and orientation of interesting points in an image [31]. These

points are counted in small patches of the image to localise the features extracted to their rough location.

To visualise this it could be considered that an image has a grid overlaid on it, each cell in the grid accumulating a histogram of gradients and orientations for all the pixels in the cell [132]. Features extracted here can then be used as part of a machine learning method by comparing the trained histogram against the histogram of an input image.

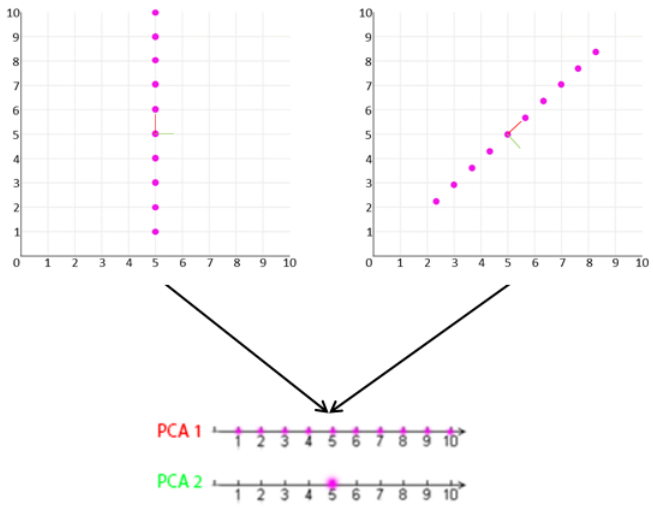
A Histogram of Orientated Gradients is a commonly used method as a step to create invariant object recognition [129] and could be considered one of the current state of the art methods along side SIFT (Section 2.2.2.6). HOG is often used with human and face detection applications, as can be seen in the literature [142], [130], and [36].

2.2.2.4 Principle Component Analysis

Principle Component Analysis (PCA) is an invariant feature extraction method that reduces the dimensionality of an image to a set of features [66]. One possible way to extract principle components from an image is to use the x and y coordinates of each ON pixel and then apply PCA to get the distribution of pixels on each axis, though a more common use of PCA with images is to represent each pixel as an axis.

Principle component analysis works by extracting important information from the matrix and producing new values called principle components [3]. Figure 2.9 shows two graphs that produce the same set of principle components, showing its invariant properties for rotation translations.

Fig. 2.9 Example of PCA



To produce principal components we create a set of Eigenvectors and Eigenvalues [32] which represent a direction in space and the variance that a point has from that directional line. Figure 2.9 shows two Eigenvectors and their associated Eigenvalues. The principle component is decided by the Eigenvector with the largest Eigenvalue.

2.2.2.5 Direction Features

Direction features work by identifying the direction of strokes in an image [140]. By identifying features as directional strokes you can also reduce the dimensionality of the image [14].

This would produce a feature that could be invariant to lighting, translation, and scale, but would struggle with rotations as it would alter the directional data of the strokes. If the directions were held in a degree of rotation from the previous stroke then it could be possible to make it invariant to rotation. With neural networks this feature would be difficult as there could be a different number of strokes and directions for each image, which would change the number of inputs into the network if each stroke is represented as a separate input. This could perhaps be solved by choosing a number of directions and counting the number of strokes for the closest direction.

2.2.2.6 Scale Invariant Feature Transform

Scale Invariant Feature Transform (SIFT) is one method for recognising images invariant to scale, translation, rotation, and a degree of illumination [90]. SIFT works by identifying a number of features in an image invariantly and storing the feature as a key or key point along with the keys orientation [85]. The key points are found using the minima and maxima of an approximation of the Laplacian of Gaussian applied to the scale space of the image [125]. In a new image the SIFT features can then be extracted and compared with the current keys on file to find a possible match, as well as observing the neighbouring keys to matches that may have been found.

SIFT is a well used method for detecting objects in images. It can invariantly detect objects on a number of possible variations, although would likely struggle with images taken from a different view where the features would seem much different. SIFT is invariant to scale, translations, and rotations [91], and due to this it is considered as one of the current state of the art methods for invariant object recognition alongside neural networks (Section 2.4).

There are many different variations and improvements to the SIFT method, including PCA-SIFT, CSIFT, GSIFT, SURF and ASIFT, which are reviewed in [139]. Of these methods each has a speciality of improving results for variations in scale, rotation, blur,

and illumination. The original SIFT and CSIFT being the best SIFT methods for scale and rotation variance in images which this thesis focuses on.

CSIFT is a variation of SIFT that introduces colour into the SIFT algorithm [2]. CSIFT can improve classification results for applications where colour plays an important role for classifying similar objects, such as apples and oranges, by building SIFT descriptors in a colour invariant space.

2.2.2.7 Edge Detection

Edge detection allows for the edges in an image to be extracted [95], allowing for a less complex image to be used for training the network. Edge detection can help to remove some of the noise in an image as well as creating an image that can be invariant to a small level of change in brightness.

To produce invariance in scale, rotation, and translation the edges that are detected could be stored as a count of the amount of edges in the image. The percentage of the image that an edge makes up could also produce invariant results in rotation and translation. Figure 2.10 shows a before and after image of edge detection for a car.

Fig. 2.10 Edge Detection

2.2.2.8 Corner Detection

Corner detection is considered a type of Edge detection [96]. Corner detection is produced by detecting two half edges that are joined to each other, where the two join is considered a corner of the object.

The corners that are detected in the image can then be used as an invariant feature by counting the number of corners in the image, as well as the size of the angle of each corner. These two features would produce results that are invariant to scaling, translation, and rotation.

2.2.2.9 Blob Detection

Blob detection is the detection of an area that is lighter/darker or a different colour to that of its surrounding pixels [33]. Initially blob detection is not invariant except for perhaps lighting, but it can be made invariant by using features such as the amount of blobs in an image, the average colour of the blobs, and the proportional scale of the blobs. These methods can create invariance in scaling, transformation, and rotation.

2.2.2.10 Ridge Detection

Ridge detection is used to locate edges in an image that would be considered a ridge. Ridge detection is often used with 3D plots[126], but could also be used to detect ridges in 2D slices. While there may be variations in the shape of the ridges, you can add some invariance by counting the amount of ridges as well as the average height of those ridges.

Ridge detection may not be very effective with certain images where the amount of ridges does not correlate to what the object is, but could be effective with other objects such as straight cut crisps, and ridge cut crisps. Ridge detection could be combined with Principle Component Analysis (Section 2.2.2.4) to detect the amount of ridges on one component mapping.

2.2.2.11 Pixel Counting

Pixel Counting is a very simple feature to extract. You count how many pixels there are of each colour in the image, and then use the result as a feature. This is invariant to rotation and transformation as the amount of pixels will not change, but can be affected by scaling and lighting as the amount of pixels would change as well as the colours.

Possible ways to overcome this would be to count the pixels as a percentage value of the image, allowing for the overall image to be scaled and the percentage of the image should stay the same, however this will not work with scaling the object inside the image. It would also be possible to have a feature that represents a range of pixels (For example, 0-125 could be white and 125-255 could be black), which can account for some variation in colour change that could alter with different lighting conditions.

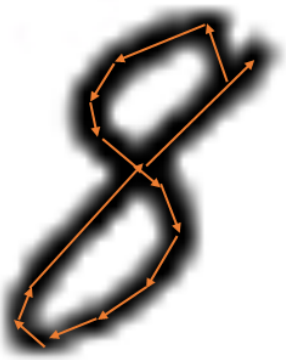
2.2.2.12 Chain-Code

Chain-code works by making a chain of pixels that link to the next successive pixel in the chain, and contain a direction indicator for what direction the stroke or chain is heading [68]. This is initially invariant to changes in lighting but not to rotation or scaling.

Chain code can be made invariant in a number of ways. To make Chain Code scale invariant you can list the proportions of each of the chains heading in each direction for each chain in the image. Chain-code could also be made rotation invariant by storing the length of a chain instead of the directions, and the amount of direction changes in a chain could also be invariant to scaling and rotation.

Combined with edge detection you could also list what edges are connected which could help reduce the error rate while also being invariant to scaling, rotation, and slight distortion (Section 2.2.2.7).

Fig. 2.11 Chain-code



2.2.2.13 Concavity

Concavity relates to the space in the cavities of an image [68]. This can be measured by working out the area of a cavity, and it's position in the image. Like many of the other feature extraction methods this is invariant to some lighting change initially but not other transformations.

Concavities can be made invariant to scale by storing the proportion of the cavity rather than an fixed area. As an image is made larger or smaller the area will change, but the proportion of the cavity in relation to the other cavities should stay the same. Concavities can also be made rotation invariant by also counting the amount of cavities there are. Regardless of what way up an image is or the size of the image there should always be the same amount of cavities.

Fig. 2.12 Concavity

2.2.2.14 Summary

Many of these feature extraction methods can be made invariant to image rotation, scaling, transformations, and lighting. They work well with simple images such as the MNIST data set of handwritten digits which contain little background noise, but with the introduction of noise and backgrounds that come with natural images, detecting features of just the object to be recognised and not the background, becomes inherently difficult and could mean that some feature extraction methods will not work at all without further image pre-processing. Feature detection methods should also be combined, as in natural images it may be found that two different objects could have the same or similar features under one type of extraction.

Feature extraction while useful for non visual datasets struggle with the noise of image data, and as such creating a network using raw pixel data may be more useful and mean less of a loss of potential useful data that you get with feature extraction. Due to this the methods listed in this thesis use raw pixel data as inputs rather than attempting complicated image processing and feature extraction. Much state of the art research emphasises on the use of raw pixel data as an input to neural networks, such as work completed by Deep Mind who create pixel-to-action autonomous learning agents [11].

2.3 Machine Vision

Machine vision is a wide field in the artificial intelligence are of research. Allowing machines to use images and video for learning applications is becoming more popular in a world with an exponentially growing amount of data. Large amounts of video footage and still images are created every day from a variety of sources such as YouTube and Security cameras. In many cases there is now too much footage generated for human operatives to manually review, and as such machines that can do the job of a human to at least mark images or clips as needing further review, can greatly reduce the workload of human operatives.

Machine vision can be conducted with a number of methods as detailed in this section as well as by using Boltzmann machines as described in Section 2.4.2.1 and Neural Networks as described in Section 2.4.

2.3.1 Image Pre-Processing

Before Machine learning is even applied to a set of images they can be pre-processed to make them more uniform for use with machine learning, reducing the difference between images which can cause lower rates of success. There are a number of techniques that can be used for this purpose which are detailed in this section. A number of these can be applied automatically to images but in some cases they must be applied manually which restricts the methods usefulness for applications to real world problems. This thesis applies some automated processing on a dataset in Chapter 3, but it can be seen in the chapter that these methods run in an autonomous fashion are not always accurate and can cause outliers in the data.

2.3.1.1 Object Centring

Object Centring is a very useful pre-processing tool for object recognition, as many machine learning methods often learn to recognise objects in the centre area of the image. In real world natural images it may be likely that the object to be identified is not centred in the image, and as such, re-aligning the object with the centre will allow for an increase in accuracy of the networks prediction.

2.3.1.2 Image Clipping

Image clipping can work well for images where the object to be identified is only a small part of the image, or the object is surrounded by a number of other objects that may interfere with the prediction process. Image clipping can work much like a humans visual perspective, focusing on an object of interests and essentially blurring out the rest of the field of view.

Fig. 2.13 A clipped image

2.3.1.3 Image Re-sizing

Many machine learning methods are dependant on having a uniform input size which should remain the same for both training and testing. Methods such as Neural Networks assign a input neuron to represent each piece of input data, and changing the input size would mean changing the network architecture. In some cases where the image is the correct size, the object inside the image may be of a different scale to that learnt by the machine learning method, so the object itself may benefit from re-sizing to match the size of objects in the training data.

When an image is re-sized it is important to retain the aspect ratio, so that the image does not become distorted and stretched. To allow for this, an image can be re-sized to the uniform size on the largest dimension, and then padded out on the other dimension to the correct size. An attempt is usually made to ensure the padding matches the background of the image, as to not interfere with the object recognition.

Fig. 2.14 A Re-sized and padded image

2.3.1.4 Image Rotation

The image can often be rotated 90 degrees due to the angle at which the camera was held to take the image. For large rotations of this size it can be obvious that the image is in the incorrect orientation, especially if the image is portrait when all the the data is landscape. Correcting a simple portrait/landscape orientation can be performed using image tool kits once identified.

Fig. 2.15 A Rotated image

2.3.1.5 Brightness and Contrast Correction

Images can often be taken in a wide variety of lighting conditions. This can make them seem a different colour to a Neural Network, which are typically trained on images with a uniform light setting. Stabilising the brightness and contract to be more uniform to the training data will help in identification of an object, as there will be less distance between the input and the learnt representation.

Fig. 2.16 A Corrected image

2.3.1.6 Noise Reduction

Noise in an image can appear in several ways, but one of the common noise sources is static noise. It gives the image a grainy feeling which can interfere with the recognition process. Static can be reduced by adding a filter to the image that can smooth pixels based on the surrounding area, although this can also cause blurring and loss of data.

Fig. 2.17 Image Noise/Static

2.3.1.7 Summary

While these are a few of the simpler pre-processing methods, there are a number of other ways of processing an image to prepare it for application to a machine learning algorithm. Even these simple corrections can be difficult, while they are easy to alter, detecting how much to alter an image correctly can be very difficult.

Other image pre-processing can include segmentation methods as defined in Section 2.3.4, such as the threshold method which can be used to remove a background from a black and white image. When pre-processing images for machine learning, where possible we would usually like to choose a method that can be automated without the need for human processing, as this can be a difficult task on a large dataset, and manually processing the data would defeat the purpose of machine learning.

2.3.2 Object Recognition

Object recognition is a key task in the machine vision field, especially in applications where objects and items in images need identifying. There are a number of different methods for recognising objects in images, including Neural Networks as defined in Section 2.4 which are used in this thesis, but also other methods such as decision trees and support vector machines defined in this section. Each method has its own advantages for object recognition which make them suited to different tasks. Some other methods discussed such as template matching and nearest neighbour algorithms are useful when applied to features extracted from images (Such as those discussed for invariant feature extraction in Section 2.2.2.2) which can also be applied to the other methods discussed in this section. SIFT is also a popular object recognition method that is invariant to scale and some rotation variance and could be considered one of the state of the art methods for object recognition (Section 2.2.2.6).

2.3.2.1 Decision Trees

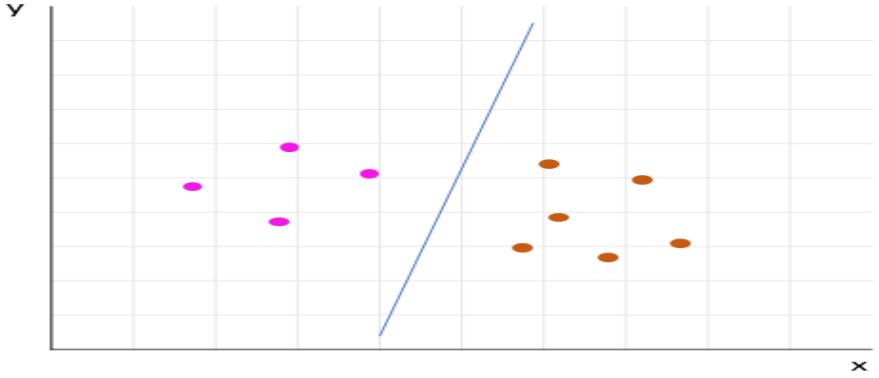
Decision trees are applicable to many situations, and while they are commonly used outside of machine vision, they can sometimes be applied to object recognition. Tasks and decisions can be carried out using decision trees, which can be considered as a set of nodes and with branches or children which represent different options of a decision. A simple example of this could be a decision tree that decides if a user should play outside, where the tree could ask a question about the weather to check if it is raining before making a decision, and can be made more complex by adding items such as "is homework completed" in the case of children. These simple trees can easily be understood by a human interpreter, but we would find trees for image/object recognition much more complicated and confusing.

Adding image variance to the object recognition problem makes a solution using decision trees even harder in cases using pixels as inputs. But in cases using invariant features as an input, a decision tree could be a useful method. For example a car will always have a set number of tyres, whether the image is rotated or not, so if those features can be extracted invariantly then decision trees could still make a decision to classify an object.

2.3.2.2 Support Vector Machines

Support Vector Machines (SVM) are useful for classification problems and can be applied to machine vision problems in a supervised manner [29]. A SVM plots data in a higher dimensional space than the one it is already represented in. SVM's then aim to find a non-linear hyper-plane that separates the different classes of data using a maximum margin method. Figure 2.18 shows an example plot of classification data on 2 axes and puts a possible plane between the two data classes. SVM's can be used as a object recognition method for images as well as 3D images [110], but as with the previous method, image invariance can make this method difficult to apply to models using image pixels as an input, and would be better applied with extracted features.

Fig. 2.18 A 2 dimensional plot and plane of a SVM represented in native space



2.3.2.3 Template Matching

Template matching is a method similar to applying filters in Convolutional Neural Networks (Section 2.4.2.3). A template is parsed across an image and a similarity is found in a similar way to pixel matching [17]. Templates are difficult to apply to images with variance in them, and as such a template can also be made for extracted invariant features to allow for invariant image recognition.

2.3.2.4 Nearest Neighbour

With invariant feature extraction a nearest neighbour classification method can be used to group objects into classifiers. One of the most common methods for this is the K-Nearest Neighbour method, which uses distances between feature vectors to group objects into a classifier [106]. Based on this an image can be sorted into a classification group with other images whose features are close together, and the label with the most images in the group is designated as the label for that classifier.

2.3.3 Image Localisation

Image localisation describes the need to identify the location of something in an image. Whether an object or face is detected in an image we will often want to know the location and size of the object in the image, and this is typically represented by a bounding box. In machine learning methods where multiple different objects are detected in an image we can output a list of objects in the image, it is much more useful for both human users and machines using generated data to know where each labelled object is rather than just its existence. Image localisation also allows for object tracking over time when a set of images are presented in series, such as tracking cars in a video for autonomous driving.

2.3.4 Image segmentation

Image segmentation can contribute to the success of object recognition in computer vision. Image segmentation refers to the process of splitting (segmenting) an image into multiple regions based on some splitting criteria. For object recognition this could be splitting an image of multiple objects into many images with singular objects in them, or segmenting an object from the image background, to assist with recognising an object with less interference.

This is particularly important in machine learning, where the objects in training images are all typically the focus and centre of an image. With Convolutional neural networks this is not as important as smaller filters are applied across parts of the image to find objects and features located in different locations of an image.

2.3.4.1 Watershed

Watershed segmentation is based on geographical segmentation from rivers and lakes, where land and plots can be segmented by some form of immediate change (such a water to grass, or fencing in fields) which indicates different segments in the landscape [116].

This idea can be used in image segmentation to separate parts of the image based on watershed lines which could indicate the perimeter of an object in an image. This can be a difficult task to perform if the object itself has many immediate changes as part of its structure.

2.3.4.2 Threshold

Threshold segmentation is a very simple type of image segmentation, especially with grey-scale images. Images are segmented by choosing a threshold value in which to separate parts of the image from the background, this is typically a pixel intensity in grey-scale images [48].

Threshold segmentation is not always perfect, and issues can be caused where the background is too similar to the object, or where the object contains colours that are the same as the background. Figure 2.19 shows some images before and after the threshold method has been applied, along with some errors that happen with batch applications of the threshold method, where in this case a characters background was different to the rest of the batch, and the main character colour was the same as the background of the rest of the batch.

Fig. 2.19 Threshold applied to images - From [10]
Some materials have been removed due to 3rd party copyright.
The unabridged version can be viewed in Lancester Library -
Coventry University.

2.3.4.3 K-Nearest Neighbour

K-Nearest Neighbour (KNN) segmentation works differently to many segmentation algorithms by looking locally at parts of the image to segment them, rather than the image as a whole [89]. Each pixel in an image is categorised separately, and then using KNN, similar pixels are grouped together to form a larger segmented chunk. KNN is a popular method for classifying numerical data and can be quite simple to calculate for data with low dimensionality.

2.3.4.4 Euclidean Distance

Euclidean Distance is similar to the KNN method, where multiple pixels are compared together to achieve larger groups of pixels that make up an object. The difference in this method is that the Euclidean Distance is calculated between two pixels, and if this is less than a certain set threshold, then the pixels are considered similar and are grouped together to create the segment [112].

2.4 Neural Networks

2.4.1 Introduction

Neural Networks are a way of learning, remembering, and classifying data. In machine learning an Artificial Neural Network is a highly simplified artificial representation of a Biological Neural Network which is made up of neurons and synapses. An artificial neural network typically consists of neurons and connections/weights between those neurons which directly affect the input/output of the neurons that the connection is between. The sum of all inputs to a neuron are put through an activation function, which provides the output for the

neuron in question. A number of networks have been developed further than basic artificial neural networks, which closer follow a biological network, such as spiking neural networks.

This chapter will cover the different types of neural networks available which are relevant to the thesis and its experiments. Other networks will also be briefly discussed with references for more information, though not explained in detail if they are not of significant importance to the thesis and its objectives.

Neural networks are one of the current state of the state of the art methods for object recognition alongside methods such as SIFT (Section 2.2.2.6). In particular Convolutional Neural Networks is one of the the most promament neural netowrk for object detection [131].

2.4.1.1 History of Neural Networks

Neural networks date back to the 1940's when Warren McCulloch and Walter Pitts presented a paper modelling how neurons of the brain work. To model this they created an electrical circuit, which became the first artificial neural network [115].

It became possible for neural networks to be created on computers in the 1950's as technology advanced. The first attempt at creating a neural network was by Nathanial Rochester who worked at IBM, and while early attempts failed, eventually a network was created [113].

A major step in the re-energising of the neural network field was the discovery of the back-propagation algorithm in the 1960's, which was applied to the neural network field in the 1970's by several researchers.

While attempts at creating neural networks died down due to limitations of technology and the application of the neural networks, a renewed interest started in the 1980's [119] when researchers created the modern networks we use today, such as multi-layer networks and Boltzmann machines.

2.4.1.2 Supervised Learning

Supervised and unsupervised learning are general concepts within machine learning that can be applied to neural networks. Supervised learning is how a majority of Neural Networks are trained. Supervised learning consists of a set of data that has been labelled with targets (i.e. What the data is or the required output). A network will predict the outcome of the data it has been given and compare it to the targets that it was given. The difference between the two is considered the error of a network, which is then used to adjust the weights of the network and essentially learn the connection between the inputs and the required output.

2.4.1.3 Unsupervised Learning

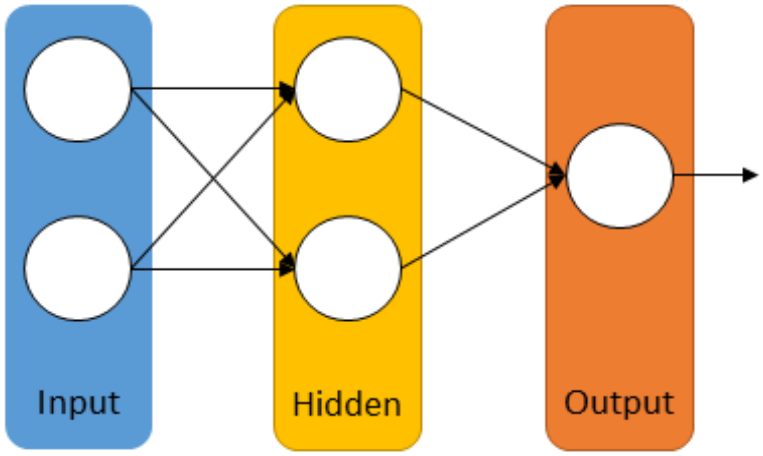
Unsupervised learning, unlike Supervised learning, does not require labelled data to learn. Unsupervised learning works by propagating the input through a network and then reversing the process to reconstruct a representation of the input. The difference between the reconstructed input and the original is considered the error value for the Network, and much like Supervised learning, it is used to update the weights and learn. Restricted Boltzmann Machines discussed in Section 2.4.2.1 are considered unsupervised networks.

2.4.1.4 Perceptrons

Perceptrons (Figure 2.20) are a very early form of a Neural Network, and are capable of recognising linearly separable data [117]. A perceptron is the simplest form of a neural network where input neurons map either directly to the output neurons or a singular hidden layer. With one hidden layer perceptrons are able to recognise non-linearly separable data, which is not possible with a direct input to output mapping.

Each connection in the network contains a weighted value, this is multiplied by the input attached to it from the input layer. All the connections to a neuron are summed and then put into a activation function to provide the neurons output.

Fig. 2.20 A Perceptron



$$y_j = f\left(\left(\sum_i w_{ij} \cdot x_i\right) + b\right) \tag{2.1}$$

A perception is trained using the perceptron rule. If the output is incorrect the perceptron rule will update the weights of the network, and will do nothing if the output was correct

[136] [50]. The Function f in the Equation 2.1 can be one of many possible functions used for neural networks, such as a sigmoid function, a step function, or a rectilinear function, which is currently a popular activation function used in neural networks. In this equation w_{ij} represents the weight between neuron i and j, x_i represents the output of neuron i, and b represents a bias.

2.4.1.5 Gradient Descent

Gradient descent is a way of tuning the weights of a neural network [63]. The idea of gradient decent is that if you were to look at the error rate of all the possible configurations of a network on a many dimension graph, you would see that across the plane of possibilities, the surface goes up and down into troughs and peaks much like a mountain range.

There can be many troughs in training a neural network, and we are looking for the one that dips the lowest. The lowest trough is called the global minima, and the rest are called local minima. Peaks would be considered maxima, with the highest being the global maxima of the network. The difficulty of this is that there is no easy way of telling if you have hit a global minima, other than if your error is constantly 0 percent. So we often get stuck with a local minima.

An analogy of this would be to imagine you are on a mountain in heavy fog without a map, you want to find the lowest valley in the space, but cannot see far around you. So when you find a valley and enter it you will not know if it is the lowest, you have to make a decision on whether to leave the valley or continue in your current valley. This is the problem with gradient decent, you can get stuck in local valleys (local minima) and it is very hard to get out.

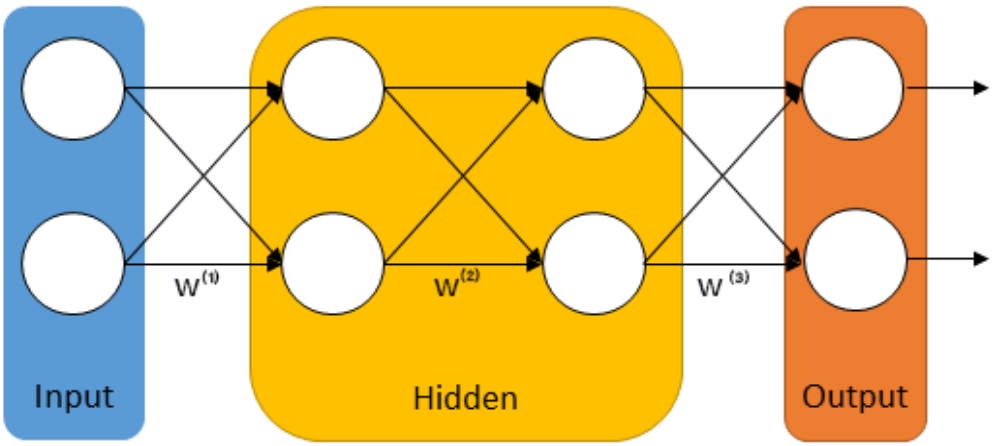
2.4.1.5.1 Back Propagation

Back propagation is used to calculate the gradient of loss in respect to the weights in a layer and distribute the error back through the network via the chain rule. A simple back propagation network can be considered as a perceptron with at least one hidden layer. Back propagation refers to the method which the network trains its weights and is capable of learning non-linear data sets [19]. A Back propagation network is a supervised network, meaning that the output is checked against a desired output before learning from any error in the prediction. Back propagation networks have been successful in many applications such as digit recognition [80] and learning other non-linear data sets [54]. The simple nature of back propagation networks also allows for the network to be altered and combined with other methods and types of networks to provide improved results [21].

The data in a back propagation network forward propagates through the network much like a perceptron, with the output value of a neuron being the activation function of the sum of each input multiplied by the correct weight. A typical activation function used for neural networks is the sigmoid function. This process continues thought the network until an output layer is reached. At the output layer, the neurons outputs are compared against the target values.

The difference between the target and the actual output provides an error value. This error value is used to calculate a neurons error and then update the weights based on that error. The error for a neuron is passed back down the network to provide an error value for the previous layer to allow each layer to update its weights in sequence.

Fig. 2.21 Back Propagation training



$$\mathbf{E} = \sum_{j} \frac{1}{2}(T_j - o_j)^2 \tag{2.2}$$

Back propagation starts by calculating the error of the output layer. Equation 2.2 shows the calculation of the error for an output layer using the squared error function. T_j represents the target value of neuron j while o_j represents the actual output value of the neuron j. Using the chain rule we need to calculate how much a weight w_{ij} (weights between node i and j) effects the total error. This is calculated using equation 2.3 where x_j is the weighted sum of the inputs to node j. The indexes i, j, and k, refer to a neuron index for the previous layer (i), the current layer (j), and the next layer (k).

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial x_j} \frac{\partial x_j}{\partial w_{ij}} \tag{2.3}$$

$$\frac{\partial x_j}{\partial w_{ij}} = o_i \tag{2.4}$$

$$\frac{\partial E}{\partial o_j} = o_j - t_j \tag{2.5}$$

Equation 2.6 represents the derivative of the activation function, which in this case the activation function f is the sigmoid activation function.

$$\frac{\partial o_j}{\partial x_j} = f(x_j) \cdot (1 - f(x_j)) \tag{2.6}$$

$$o_j = f(x_j + b) \tag{2.7}$$

$$x_j = \sum_i o_i w_{ij} \tag{2.8}$$

Equation 2.7 shows how the output of a node j is calculated where f is the activation function and b is the bias. Equation 2.8 shows the calculation of x_j where index i is a node on the previous layer. Equation 2.9 shows the weight update equation to calculate the new weight based on the calculated error. α represents the learning rate of the network.

$$w_{ij} = w_{ij} - \alpha \frac{\partial E}{\partial w_{ij}} \tag{2.9}$$

The back propagation process for the hidden layers in the neural network is similar to the output layer, however the difference here is that the error value produced by the difference between the target and output has been replaced with an error value (delta) passed back from the layer above. Equation 2.10 and 2.11 show an altered calculation for a hidden layer. o_k represents the output of a neuron index k on the next layer. $\frac{\partial E}{\partial o_j}$ and $\frac{\partial E}{\partial o_k}$ in these equations are often referred to as the delta.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}} \tag{2.10}$$

$$\frac{\partial E}{\partial o_j} = \sum_k w_{jk} f(x_k)(1 - f(x_k)) \frac{\partial E}{\partial o_k} \tag{2.11}$$

This training is performed for every item in the dataset and then repeated multiple times. Each iteration of the dataset is called an epoch. The training is usually stopped when the network is no longer improving its error rate, or it has learnt how to exactly represent the data, or a hard limit in epochs has been reached.

Different forms of neural networks are used to solve the objectives of this thesis in the experiment chapters. The black box nature of neural networks make them a suitable machine learning method for solving the problem of creating an invariant machine learning method that does not need training on the different variations of training data.

2.4.1.5.2 Real Time Learning Networks

Many neural networks are trained on a pre-defined dataset before the trained network is implemented in a real world application. These networks can be updated by running further training and passing an updated weight matrix to the application. This can be useful but means that when a network comes across something new it will not be able to identify it until it has been classified and re-trained.

This can be solved with networks that learn in real time while they are being used. Typically reinforcement learning and recurrent neural networks (Section 2.4.2.4) are networks that can learn well in real time, but also basic neural networks can learn by learning by new images it comes across to reinforce a strong classification. Because this is unsupervised it can also cause issues where the machine learning algorithm reinforces an incorrect assumption, and in some networks it could cause it to forget previously trained functions over time.

A number of robotic applications learn new items in real time [98], such as ASIMO by Honda [5], which can learn new objects by being shown them, and then classifying the new object after prompts from human speech, much like how a child learns about new objects from their parents. Learning in real time will become more important in a quickly changing world, and become closer to real adaptive artificial intelligence similar to that of human intelligence. Control systems also need to learn and adapt while also maintaining its normal functioning, presenting another example of using real time learning [67].

While real time neural networks are not used in this thesis, they are an interesting topic that could expand the methods introduced to allow them to be used with real time data in applications such as robotics.

2.4.2 Types of Neural Network

There are many different types of neural networks that are made up of neurons but connected in different ways and use different learning algorithms. Each of these networks have their own advantages and applications which they are suited to, such as still images, or sequences of data over time such as voice.

2.4.2.1 Boltzmann Machines

Boltzmann Machines are a type of unsupervised network that are similar to back propagation neural networks (Section 2.4.1.5.1) in structure. Boltzmann machines train their weights by reconstructing their input and comparing it to the original input allowing the neurons to calculate an error value which they can then use to update the layers weights. Unsupervised methods such as Boltzmann Machines are useful for initialising weights for supervised deep neural networks which would otherwise need to initialise weights with random values, and are used as such in Chapters 3 and 4.

A Boltzmann machine consists of random variables $\mathbf{v}, \mathbf{h}$ where the probability of the two variables can be calculated using an energy function as below.

$$P(\mathbf{v}, \mathbf{h}) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{Z} \tag{2.12}$$

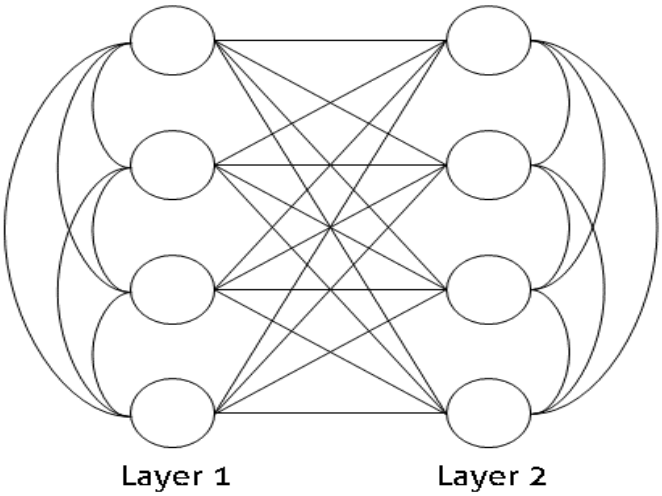
where Z is a normalisation factor

$$Z = \sum_{\mathbf{h}} \sum_{\mathbf{v}} \exp(-E(\mathbf{v}, \mathbf{h})) \tag{2.13}$$

A Restricted Boltzmann Machine has the following energy function [35]:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i} v_i b_i - \sum_{j} h_j c_j - \sum_{ij} v_i W_{ij} h_j \tag{2.14}$$

Fig. 2.22 A Boltzmann Machine



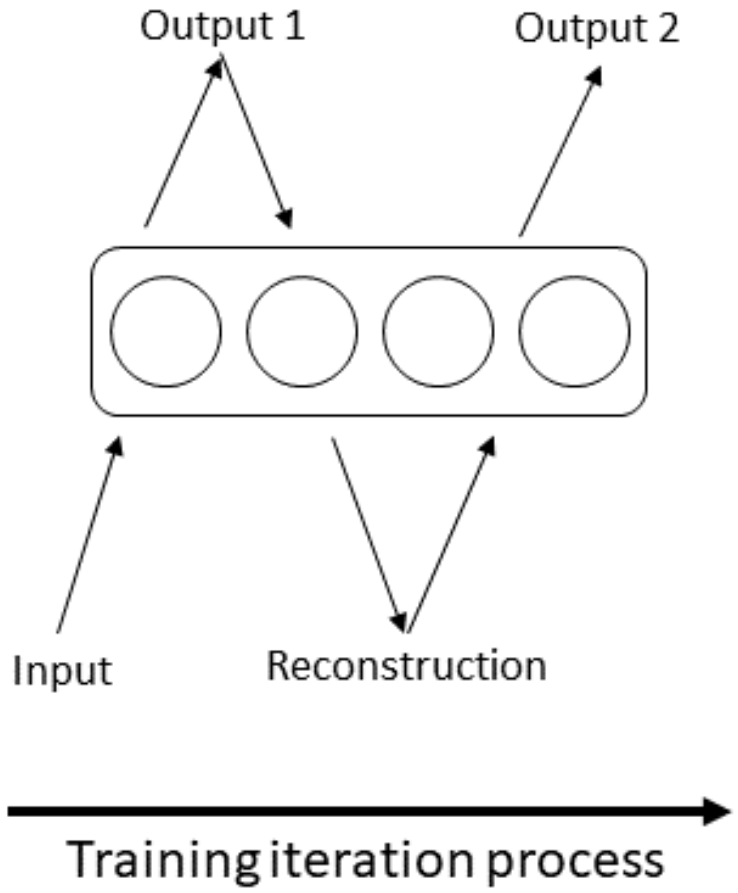
A Restricted Boltzmann Machine (RBM) is a Boltzmann machine with additional constraints. A RBM is constrained by limiting connections between neurons so that connections can only be between neurons in different layers and not between neurons inside the same layer [22], whereas a normal Boltzmann machine may have connections to other neurons in the same layer.

A RBM is trained by forward propagating the input through the Boltzmann machine. The output is then passed back though the Boltzmann machine, essentially forward propagating in the reverse direction. The returned results represent a reconstruction of what the network thinks was input into the system. This reconstruction is then fed-forward and compared to the original output to produce an error value to allow for weight updates.

The error value is then used to update the weights of the Boltzmann machine. In a stack of Restricted Boltzmann Machines this training is done one layer/RBM at a time. So the first layer will have all the input data run though it over a number of epochs and before then moving onto training the next layer/RBM.

Equation 2.15 shows the weight update equation for a Restricted Boltzmann Machine. w_t represents the weight matrix for an RBM with w_{t-1} representing the weight matrix state before iteration t. o_1 and o_2 represent the RBM output on the first pass with the input and the second pass with the reconstructed data, while x_1 represents the the original training input and x_2 represents the reconstructed input. $\times$ in this equation represents the cross product of

Fig. 2.23 A Restricted Boltzmann Machine during training



the matrices. Figure 2.23 shows a visualisation of a training iteration of an RBM which is essentially a Markov chain from Gibbs sampling limited to a couple of iterations [120].

$$w_t = w_{t-1} + \alpha((o_1 \times x1) - (o_2 \times x_2)) \tag{2.15}$$

Boltzmann machines have two bias values corresponding to a bias for visible units (c) and a bias for hidden units (b). The weight update for these bias units are represented by Equation 2.16 and 2.17.

$$c_t = c_{t-1} + \alpha(x_1 - x_2) \tag{2.16}$$

$$b_t = b_{t-1} + \alpha(o_1 - o_2) \tag{2.17}$$

In a stack of Restricted Boltzmann Machines, once all the machines in the stack are trained, the weights can then be collected and given to a supervised neural network with the same architecture, to initialise the networks weights for back propagation training. An RBM is suitable for pre-training a neural network as the architecture of a stack of Restricted Boltzmann Machines is similar to that of a neural network, with the same neuron and connection architecture between layers and an energy function where the probability of an output node being 1 is the same as an activation function for a neuron in a neural network. It is not necessary to pre-train the output layer of a neural network using an RBM, as the output layer represents the classifications and not features of the image. Training the output layer using an RBM can actually cause results of a pre-trained network to become significantly worse.

An RBM on its own can be used to generate images of what neurons have learnt, by setting the desired target output on the output layer, you can feed back up the layers until you are given a representation of the input required to obtain the desired output. This image gives a visual representation of what the network has learnt to recognise.

Restricted Boltzmann Machines can be used for a number of purposes such as phone recognition [30] and image recognition. RBM's can also be used for pre-training deep neural networks as detailed in Section 2.4.4.1.

Restricted Boltzmann Machines are used in Chapters 3 and 4 as pre-training networks to initialise the weights for some deep fully connected networks.

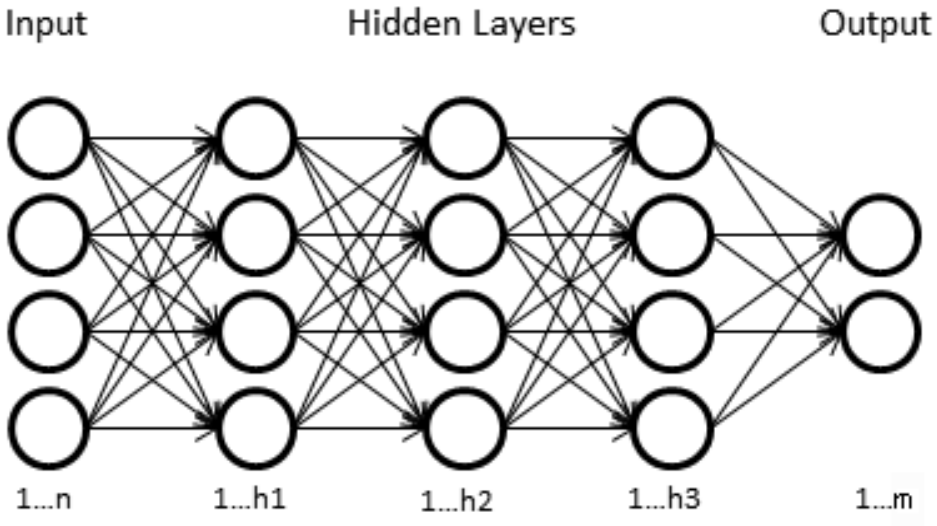
2.4.2.2 Deep Neural Networks

Recent developments in machine learning, known as 'Deep Learning', have shown how hierarchies of features and higher-level representations (e.g. object parts) can be learnt in an unsupervised manner directly from data [63]. Deep learning has shown some promising results that show that it would be a good method of attempting to solve the invariance problem [143].

Deep Learning Architectures have attracted a significant research interest in recent years. These are architectures with a larger number of hidden layers [62]. Hinton introduced an unsupervised, fast, greedy learning algorithm that finds a fairly good set of parameters quickly in deep networks with millions of parameters and many hidden layers. The unsupervised learning in the Deep Architectures enables extracting salient information about the input distribution captured in the hidden layers [12]. The features extracted in the first hidden layer are seen as low-level features. These are then fed as the input in the second layer where the new extracted features are higher-level features [12]. This approach leads to steadily higher-level abstractions of the training data, and the potential to learn complex relationships with much fewer neurons than a network with only one hidden layer would be able to [81]. Deep architectures have seen use in a number of application areas, for example object recognition [74], speech recognition [61], and musical genre categorisation [141].

Deep Neural Networks are a type of Neural Network containing many hidden layers between the input and output layers of a neural network, typically more than one or two layers [12].The layout of a Deep Network is shown in figure 2.24. The figure shows 3 hidden layers sandwiched between the input and output layer, and each hidden layer may contain a different number of neurons.

Fig. 2.24 A sample layout of a Deep Network.



Deep neural networks, unlike shallow neural networks, require the correct initialisation of weights in order for the network to be able to work efficiently with the large amount of connections it has. One possible way of initialising the weights is though pre-training a deep network using Restricted Boltzmann Machines as detailed in Section 2.4.4.1.

Deep learning plays an important part of this thesis as all methods and applications presented in Chapters 3, 4, 5, and 5.8 make use of deep learning.

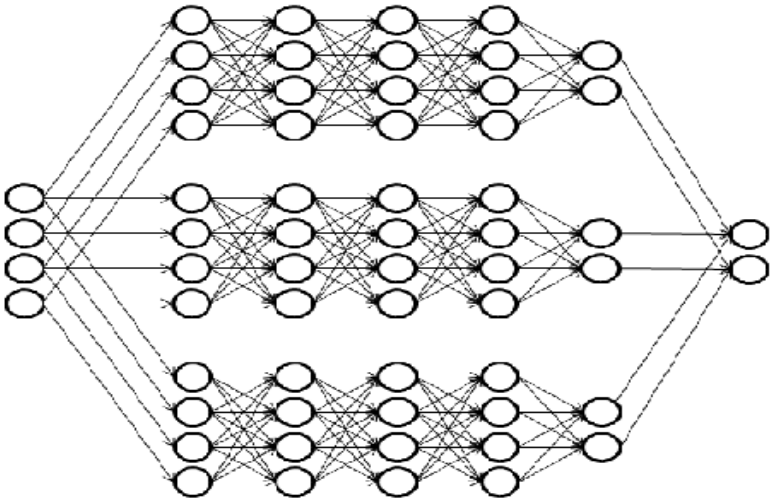
2.4.2.2.1 Multi-Column

Multi-column deep networks are a way of combining more than one network together into one model [27]. Each neural network becomes a column of the multi-column network, with each network receiving the same input. The results of each network are then combined to find the average result for each output neuron.

The use of a multi-column network can produce favourable results when compared to the accuracy of other neural network methods against well known data sets like MINST, NIST SD 19, CIFAR10 and NORB [27].

Multi-column networks have been used previously for applications such as traffic sign recognition [24] and Chinese character recognition [25].

Fig. 2.25 A sample layout of a Multi-Column Network.



Multi-Column networks have helped to inspire the invariant method in Chapter 5, where the convolutional network splits the convolutional layers into columns to allow each column to apply filters in a different rotation, however the combining of the results are performed with pooling rather than combining the output neurons on each column to a separate output layer.

2.4.2.2.2 Deep Convolutional Neural Networks

Deep Convolutional Neural Networks are an extension of Convolutional Neural Networks covered in Section 2.4.2.3. Deep Convolutional networks can have many more filter layers as well as having a deep neural network attached to the output.

The advantages provided by using a deep architecture with convolutional networks allow for complex data sets such as Image-net to be used which is based on natural images [75], or other complex visual recognition challenges [40].

2.4.2.3 Convolutional Neural Networks

Convolutional neural networks (CNN) are a very unique type of neural network when compared to other types of networks. Convolutions networks are a powerful tool for recognising objects in images where patterns may re-occur in multiple positions in an image, which is typically the case with natural images, especially if they contain multiple of a similar object. This is done though a number of filters that have learnt to recognise a particular pattern or feature, which can then be applied many times to the different parts of an image. A CNN can also help provide some invariance to shift and distortion [82] due to the filter being

applied across all positions in the image. Neural Networks also struggle with scaling to larger images, or images with multiple channels, which CNN's can scale much better [70] as they are designed to expect images, whereas Neural Nets are designed for a variety of types of data. An advantage of Convolutional Networks is that they require fewer weights than a fully connected layer due to weight tying between neurons.

CNN's have been used to improve results of recognition in applications such as hand-written digit recognition [26], video quality assessment [79], and face recognition [78]. This thesis makes use of CNN's as part of the core contribution chapters for creating a rotation invariant network in Chapters 5 and 5.8.

Convolutional Neural Networks are typically made up of a number of different layers. This includes Convolutional layers, pooling layers, and fully connected layers. Pooling layers are usually applied after convolutional layers to reduce the size of the data before the next layer, and fully connected layers are usually found on the last steps of a network after all convolutional layers have been completed to allow for classifications to be learnt. Rectified Linear Unit layers are applied after a convolutional layer to apply an element wise activation function [70] to the output of a convolutional layer.

Inside a convolutional layer we find a 4 dimensional structure of weights compared to the 2 dimensional structure of traditional Neural Networks. The first dimension of this structure relates to the number of filters in the convolutional layer, the second dimension relates to the depth of the filter, while the third and fourth dimensions are the height and width of the filter. Each filter in a convolutional layer is applied multiple times to each point in the layer input, shifting the filter across the input until it has been applied to the entire input, the amount a filter is shifted each time is often called the stride, which typically starts at 1 (applying the filter in every possible combination it its across an input) and can be increased to reduce the size of the layer output.

Typically convolutional layers are thought of as 3 dimensional structures as neurons inside a Convolutional layer are connected in 3 dimensions (height, width, and depth), while neurons for different filters are not connected. The 4th dimension mentioned above is used to differentiate between filters in a weight matrix. Figure 2.26 shows part of a CNN with 3D input and 3D output and Figure 2.27 shows an example of the application of a filter to part an input (such as an image). A filters depth will usually match the number of channels on the input data.

Fig. 2.26 Part of a Convolutional Network - From [70]

Fig. 2.27 Applying a filter - From [70]

Figure 2.28 shows a more whole convolutional neural network. This network takes an image as an input and runs multiple pairs of convolutional and pooling layers, before ending in a fully connected neural network for classification. Pooling layers can take many forms, from average pooling to max pooling, with max pooling being the typical pooling used. Max pooling works by combining a set of values into one by representing them with the highest value of the originals, for example pooling a 2 by 2 grid into one value where the grid has the values 5, 2, 3, and 7 the output from the pool representing this data would be 7.

Fig. 2.28 A Convolutional Network - From [28]

Some materials have been removed due to 3rd party copyright. The unabridged version can be viewed in Lancester Library - Coventry University.

At the end of the convolutional networks there are a set of fully connected layers [124], acting as a typical back propagation network. We can train this network using a back propagation algorithm in the normal manner, with the error rate for each neuron being passed up each layer to its connected neurons. Where neurons have been pooled in pooling layers the error value can either be simply duplicated and passed to each neuron that was combined in the pool or more commonly the error value is passed only to the neuron that won the pool with the remaining neurons not learning.

A great example of a CNN for object recognition is its use with the ImageNet dataset [74], which combined with deep learning achieved an error rate of 15.3% in the ILSVRC-2012 challenge for top-5 test error rate, which is a great achievement given the data is split into 1000 categories.

Convolutional Neural Networks are designed to work well with machine vision problems and as such are the ideal neural network to use for solving the rotation variance problem. The filters in Convolutional Neural Networks are an ideal method to manipulate to provide an invariant machine learning method as discussed in Chapter 5.

2.4.2.3.1 Forward Propagating a Convolutional Layer

To forward propagate a Convolutional layer using matrix multiplication the input first needs to be converted into columns that make up all the different patches that filters are being applied to in an input [70]. This is an intensive step in terms of processing and memory due to the duplication of the input data where the application of filters overlap. As mentioned in the previous section, increasing the stride reduces the amount of overlap between these columns but also reduces the amount of shift the convolutional network can deal with in locating patterns in the input. The converted columns are stored in a two dimensional matrix with the first dimension the size of the column and the second dimension being the number of columns.

The weights of a convolutional layer are converted into a 2 dimensional matrix, where the amount of filters is the first dimension and the second dimension is the product of the height, width, and depth of the filters[70].

With the two matrices a matrix multiplication can be conducted to give the dot product between each filter and every input column of data. The output of all filters is then reshaped into a 3 dimensional output with the first two dimensions corresponding to the number of shifts of the filter in the x and y dimension, and the third dimension equalling the number of filters [70].

2.4.2.3.2 Backward Propagating a Convolutional Layer

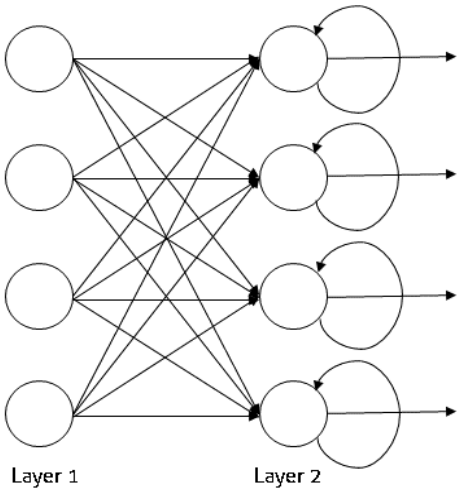
During back propagation the chain rule is applied to calculate the gradient component for each weight and calculating the error using the deltas for the layer [52]. This is a similar process to back propagation where the convolutions are undertaken in reverse using the deltas calculated for each neuron in a filter.

2.4.2.4 Recurrent Neural Networks

Recurrent Neural Networks are a type of neural network that form a circular architecture. Outputs from the nodes in the network can be used as inputs to other neurons from previous layers or the same layer.

Recurrent networks can be used in applications where a network needs to be run in real time, where it is constantly learning and predicting, while also remembering information that it has been taught [137]. The networks can also be used for other applications of neural networks as described in literature [76][105][107] [6].

Fig. 2.29 A possible Recurrent Network Configuration



2.4.2.5 Fuzzy Networks

Fuzzy Networks combine the use of fuzzy rule based systems and neural networks. A Fuzzy Network uses the rules of a fuzzy system as a replacement for the activation functions of neurons. The weights of the network can then be trained to improve accuracy of the rules of the fuzzy system [18]. This type of hybrid system requires some sample data to compare inputs and outputs to find any error in the fuzzy network, but reduces the need for a human operator to fine tune the system which is typically the case with fuzzy systems.

Literature [94] [84] [23] shows several different applications and methods for using fuzzy systems, such as stock trading [77].

2.4.3 Hyperparameters

Neural Networks have many different hyperparameters that control how large a network is, the batch size of a network, and the learning rate for a network. Extra methods used with neural networks such as dropout also have their own set of hyperparameters such as the drop rate for each layer, or the number of buckets used for data that is being bucketed.

Each of these parameters are difficult to choose as there are no definitive methods to selecting the correct hyperparameters for the problem at hand [8]. This is usually solved through trial and error to find an optimum set of hyperparameters that work together, but there are also some methods that will try and intelligently choose the correct hyperparameter, such as network growing and pruning for the network architecture (Section 2.4.4.7).

Selecting the correct hyperparameters is an important task for neural networks. Selecting bad parameters usually means a large drop in network accuracy compared to networks with well selected parameters. And fine tuning these parameters can improve a network greatly before Neural Network improvements such as dropout are even applied.

Some machine learning tool-kits such as Tensorflow [1] have built in functions to search for optimum hyperparameters through trial and error automatically. The functions take a range or list for each parameter being tuned and will run a number of experiments using different variations of those parameters together.

The experiments in this thesis use a mixture of trial and error to find optimum hyperparameters, as well as using a number of existing and published architectures for the problem being solved, however a large scale search for the most appropriate hyperparameters would be difficult due to the run-time of the method being investigated.

2.4.4 Improving Neural Networks

While neural networks preform well at their intended tasks, there are many methods that can be applied to a network to improve the way it functions. These methods can yield results from an increase in speed to a reduction in error of the network.

This section will give an overview of some of these methods and an introduction to how they work and the expected outcomes of introducing the methods to the neural networks that we will be using. A variation of these methods are used in different experiment chapters in order to improve results of the different experiments being conducted.

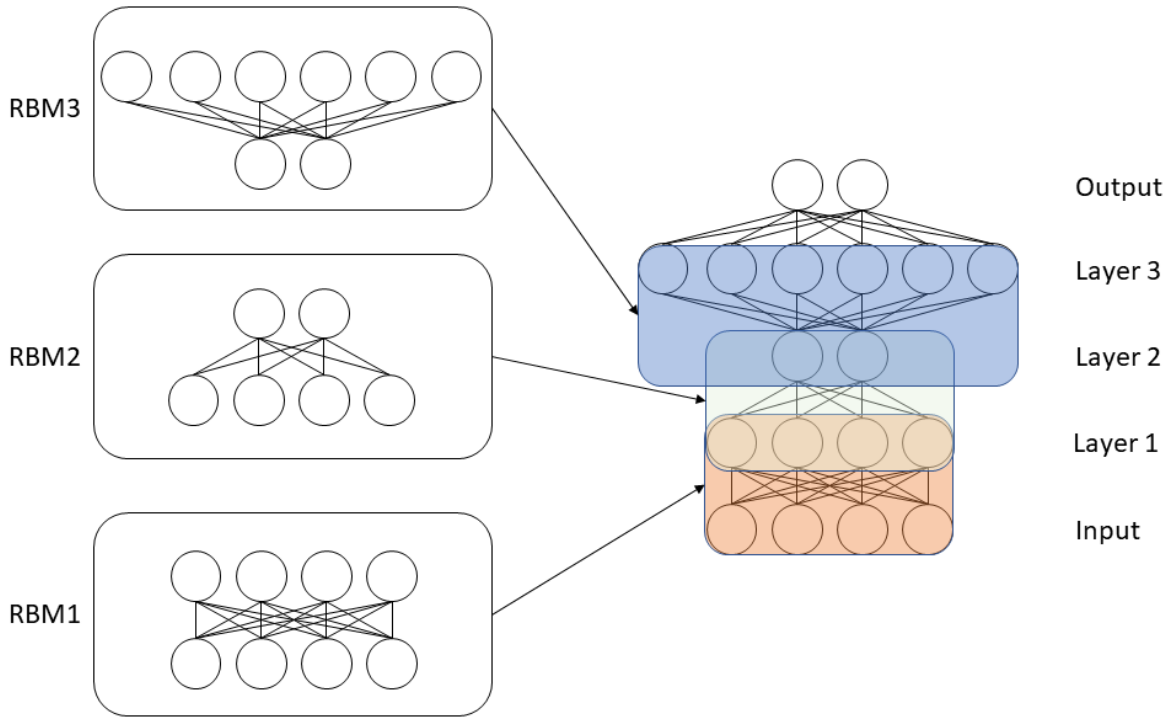
2.4.4.1 Pre-Training

Deep Neural Networks are difficult to train when more than 2 hidden layers are present. A method of pre-training the weights of a Deep Network allows for the weights to be initialised close to the weights that will be required for recognition. This is done using a Restricted Boltzmann Machine [62], which starts with randomly initialised weights and is then trained using unsupervised learning. To train this RBM an image is passed through the machine to get an output, this output is then fed back through the RBM to get a reconstruction which can then be forward propagated again to obtain a second output. These two outputs are used to calculate the weight update for the RBM as described in Section 2.4.2.1.

This training happens for each RBM in a stack representing an individual layer in the network, with training iterating over many epochs for each RBM before continuing to train the next RBM in the stack. The input to RBMs further up the stack are passed through the previously trained RBM just like data travelling through previous layers in a neural network.

Once trained, these weights are then introduced into a Back Propagation Neural Network with the same architecture for supervised training, the result of this network can then be used for prediction. Figure 2.30 shows a stack of RBM's that represent each layer in a deep back propagation network.

Fig. 2.30 Multiple Restricted Boltzmann Machines used for pre-training a Deep Neural Network



Pre-training using Restricted Boltzmann Machines is performed in Chapter 3 and 4 to initialise weights for the plain deep neural networks.

2.4.4.2 Regularisation

Regularisation causes networks to favour having smaller weights where possible, allowing larger weights only when they have a significant positive impact on the network [99], this applies not only to weights but also to training time and the size of a network. This regularisation allows for the network to be kept more general rather than overfitting a training set, allowing it to more accurately predict on data the network has never seen before. There are several ways to regularise a network with several improvements in the following sections falling under the umbrella of regularisation, such as early stopping (Section 2.4.4.4) which could be seen as regularisation over time. Regularisation can also be performed by using

the correct number of hidden nodes and layers, which can be difficult to determine, with network pruning and growing helping with this technique (Section 2.4.4.7). Weight decay as mentioned in back propagation variations (Section 2.4.4.3) is a method of regularising weights by keeping them small unless they are particularly useful to the network.

Regularisation is important in Chapter 5 due to the growing size of weights combined from the multiple parallel layers, and is performed through the use of Batch Normalisation (Section 2.4.4.6).

2.4.4.3 Back Propagation Variations

There are several variations of back propagation that can help improve the methods accuracy and efficiency. Some modifications include momentum, weight decay, and cross entropy [13]. Momentum is a method to prevent getting stuck in local minima by providing a value similar to the learning rate which allows the network to jump out of local minima, where the momentum value directly effects the size of the minima trench that can be jumped out of. This value is a portion of the previous update and is added to the current update.

Weight decay decreases a weight by a proportion during the update process of back propagation. This has the effect of pushing weights which are not learning or updating towards 0 and causing back propagation to continually reinforce useful weights. Cross entropy is designed to reduce learning slowdown [99] by introducing a new cost function. Cross entropy is useful when nodes are saturated as it allows errors to still alter weights in the network [108].

2.4.4.4 Early Stopping

Early stopping is a method introduced to neural networks to prevent over-fitting. Rather simply the method involves stopping a network before its Hard-Stop time when it is detected that the network has not learnt anything new over a set number of iterations.

Early stopping is usually applied with a validation set [111]. The validation set is tested on the network along side the training set during training, and when this validation set has not gained an improved accuracy for a number of iterations while the training set is still learning, is when over-fitting is considered to have happened. At this point we stop training the network and take the result from the snapshot with the highest result on the validation set.

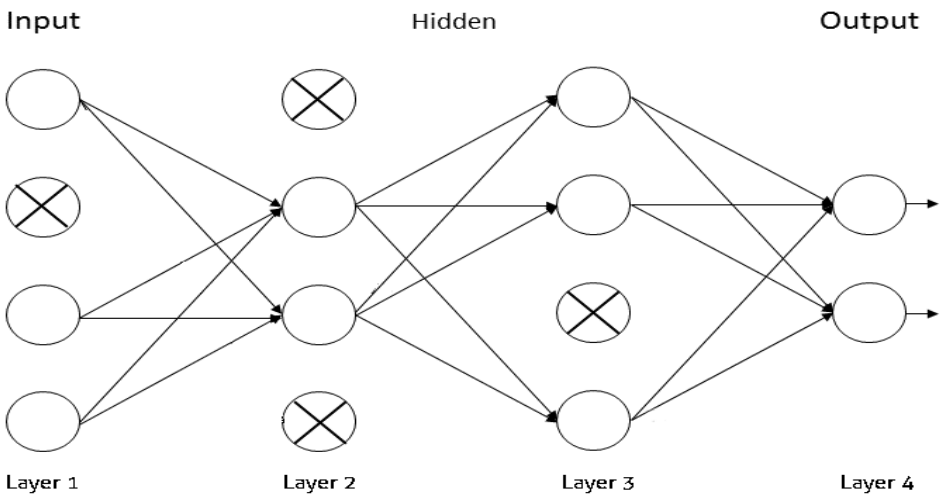
2.4.4.5 Dropout

Dropout is a relatively new area in the field of Neural networks and has been applied to the area of deep learning [128]. Dropout helps to prevent the over-fitting of a neural network

by switching off neurons randomly during a training epoch. Dropout has been successfully applied to the ImageNet data set [75] to provide an increase in accuracy over the default neural network.

Dropout prevents over fitting by forcing neurons that may otherwise be underused to learn new features and representations. This is achieved by turning off random neurons during training, causing the features learnt by those neurons to no longer be available to the network temporarily, and as such the other neurons that were previously underused may learn to replace the lost data caused by other neurons being switched off during training. This can either reinforce current detected features, or cause the network to pick up new features that were not recognised before.

Fig. 2.31 A Dropout Network during training



To perform dropout a number of neurons are switched off from each layer, including the input layer but excluding the output layer. Once these neurons are selected we ignore them and their associated weights. We can then continue to train our network by forward propagating and back propagating without these values.

The neurons that get switched off are selected by giving each neuron a probability to be dropped, which is usually the same for every neuron in a layer, and then comparing that probability to a random number. A 50% probability for each neuron in the layer will usually mean 50% of the neurons in that layer are switched off, but there is a small probability slightly more or less can be switched off.

At the end of a training epoch we can then turn these neurons back on, and if required select another set of neurons to turn off for the next epoch. A simple way of turning off

neurons in a network is to set their associated weights between each layer to zero, and then recall their weights at the end of the epoch.

During forward propagation after training, neurons outputs are reduced in size relative to the amount of neurons that were turned off during during training with dropout active to account for the increase of input to a neuron when the previous layer is all switched on. If $\mathbf{x}^{(l)}$ represents the node outputs at layer l, and p_l is the proportion of nodes retained during dropout for layer l, the node output during prediction would be:

$$x^{(l)} = sigmoid(p_l W^{(l)} x^{(l-1)} + b) \tag{2.18}$$

Which accounts for the increased number of input nodes when dropout is switched off compared to the number of input nodes when dropout is active.

Chapter 4 discusses a method called selective dropout, which allows for neurons to be dropped by a calculated statistic instead of being randomly dropped as it is with traditional dropout. With this method it is also still possible to choose a drop probability for a layer so that all neurons probabilities still average to the selected drop proportion.

2.4.4.6 Batch Normalisation

Batch Normalisation [64] is a relatively new method designed to dramatically improve overfitting issues as well as accelerating the rate at which a neural network can learn. Neural networks typically have to use small learn rates due to a layers inputs changing each iteration as the previous layers learn. Ioffe and Szegedy [64] introduce this phenomenon as 'internal covariate shift".

Batch normalisation allows for larger learning rates by normalising layer inputs for each data mini-batch to reduce this covariate shift. This also often regularises data removing the need for improvements such as dropout for reducing network overfitting. Figure 2.32 shows how the data in a batch are normalised using the mean and the variance of the input batch. A linear function is used to scale and shift the data so that variance is equal to γ or β. ϵ in Figure 2.32 as part of the normalise equation is a very small constant value that prevents dividing by zero if the variance happens to be zero which rarely happens.

Fig. 2.32 Batch Normalisation Equations - From paper [64]

During back propagation the loss/delta needs passing back through the batch normalisation method in order to pass correctly scaled loss data to the previous layer. This is done through the chain rule following the batch normalisation steps in reverse [86].

Batch normalisation is used for methods presented in Chapter 5 as Batch Normalisation has shown to be more effective than dropout while also reducing the number of iterations needed to converge a neural network.

2.4.4.7 Network Growing and Pruning

One of the biggest issues when running a neural network is initialising the network with the correct architecture and structure. Too many layers and neurons can cause the network to become more complex but at the same time over fit the training data, too few neurons and there is not enough neurons to be able to correctly identify patterns in the data. Two ways around this are Growing neural networks and pruning neural networks. As the names suggest they both do the opposite of each other and cause the network to change in size.

The idea of growing neural networks is to add extra neurons to the hidden layers of a network during training. The network is trained until a threshold has been reached in the error rate or the network is no longer learning, and then an extra neuron is added to the hidden layers and the process repeated. Once a neuron is added and there is no noticeable improvement during the training the optimum size of network has been found [101].

Network pruning is done in a similar fashion to growing networks, except the network is initialised with a very large architecture and after each training period some neurons are

removed. It is possible to do this randomly, but could cause the network to lose useful data and forget features it has learnt. So where possible we can try and ascertain which weights are the most and least important and remove the least important ones [101]. A number of different methods can be used to find a heuristic that can be applied to identify if a neuron is least important, however as it is a difficult task these methods are not always optimal. One method for this is to monitor the changes in the weight during training or by how much the error for each neuron changes during training, similar to a statistic used in the selective dropout method (Chapter 4).

Selective dropout presented in Chapter 4 achieves a result that may be considered similar to network pruning due to the switching off of neurons considered to not have learnt useful information, however only temporarily during training.

2.5 Related Work

Invariant object detection is a complex field in the artificial intelligence area due to an almost infinite number of permutations across all the different types of variance that relate to computer vision.

This section looks at a number of existing research works in the field of invariant object detection for neural networks as well as machine learning in general. Methods can range from traditional brute force (Section 2.2.2.1) to feature extraction (Section 2.2.2.2) and other more intelligent methods such as training extra networks to learn about variance.

2.5.1 Higher Order Neural Networks

Higher Order Neural Networks are similar to feed forward neural networks, however the input to the network is transformed to include terms which are products of multiple input values [43]. Higher Order Neural Networks have some problems that cause them to be inefficient and undesirable as a method due to the explosion of the number of weights required for the neural network.

To produce a Higher Order Network with invariant outputs, it is possible to 'handcraft' neurons to become invariant to transformations of input. Methods include averaging the weight matrix removing outliers that interfere with invariant recognition [53]. Handcrafting neurons can be time consuming and can take away from the nature of being able to train a neural network without needing to interfere with the training process, and as such would be an inappropriate method for creating a rotation invariant neural network with the aim of minimal dataset and network fine-tuning or manipulation.

2.5.2 Face Recognition

Face recognition is another key task in the machine vision field that is worthy of its own section. Similar to object recognition, face recognition is not only used to detect faces in images in a generalised format, but to also identify unique faces for security applications where face recognition is used. Work from this thesis is just as applicable to face recognition as it is object detection, and would be helpful in identifying facial features that may be presented at different orientations.

Many machine learning methods make attempts at facial recognition, including neural networks (Section 2.4) and Support Vector Machines [59] (Section 2.3.2.2). Many applications are possible for facial recognition algorithms including those used by large companies

such as Facebook as discussed in Section 1.1.4. Similar problems that are found in object recognition are also found in face recognition, such as lighting and pose.

A popular method for identifying faces is a Histogram of Orientated Gradients (Section 2.2.2.3) [122]. HOG extracts basic facial structures as lines and directions removing a great deal of unnecessary data. This histogram can then be compared to a histogram made from a combination of histograms for a large number of faces which should show a large similarity in the location and direction of the facial features.

A large problem with face recognition is pose, as typically faces can be presented at different rotations in each 3 dimension plane, such as a tilted head, a head looking up or down, and a head looking left or right. Some solutions to this translation problem are made by using facial landmarks which do not change with different head poses and identifying faces from the landmarks using regression trees [122]. Lighting is also a large problem as many facial recognition applications are now being used with mobile phones as a way to unlock the device, where users make use of the device at many different times of day in different lighting conditions.

2.5.2.1 Rotation Invariant Face Detection

Multiple networks can be used to detect a face invariant to rotation as explained in the literature [118], by having a router network and a set of detector networks. An image is split up into a set of overlapping smaller images, much like a convolutional neural network, and then applied first to the router network. The router network has been trained to output the orientation of the face in the image and will give a meaningless result for other objects. Using the orientation outputted by the router network the image can then be rotated so that the possible face is upright, before being applied to a set of detector networks which will identify if the image contains a face or not. The literature also identifies the importance of training the detector networks with non face images to allow it to differentiate between faces and non faces, allowing it to achieve 79.6% accuacy over two large datasets of faces.

The idea of a router network to detect the rotation of an image is an intriguing idea. Section 2.3.1 discussed the different simple ways of editing an image before we use them, using a router network like discussed in the literature could provide an efficient way to correct a image before it is put through a detector network. If a set of parallel networks could be made to detect the focus of an image, the rotation, scale, and position, then each image could become uniform enough to increase the chances of a correct detection. Routing networks follow a similar idea to that of mental rotation as described in Section 2.5.2.2.

2.5.2.2 Mental Rotation

Mental rotation is the idea that when humans perform object recognition they mentally rotate the image into an upright position for object recognition [45][133]. This would indicate the human visual system can detect the orientation of an object and the orientation that the object should be in to be upright. This would biologically support a routing network as discussed in Section 2.5.2.1 that could identify whether an object is in the correct orientation and then correct it before detection.

2.5.3 Bio-Inspired Learning

Neural Networks originate from trying to replicate the process that occurs in humans and animals in processing data very quickly and accurately. Many successful applications of technology to real world problems have been formed by studying and replicating solutions found by nature elsewhere. This gives cause to create learning algorithms that replicate the way that some Bio-organisms learn and act, as to more accurately learn how to represent objects. Typically neural networks are a very simplified representation of a human brain and as such more types of networks have appeared that try to closer replicate the brain process, such as spiking neural networks.

Spiking neural networks [134] make use of the phenomenon where chemicals build up in neurons and synapses in the brain before hitting a threshold and spiking to the next connection. The simplest form of spiking neural networks use threshold firing where the input builds up and the neuron only activates when a threshold is reached.

A development of Spiking Neural Networks, called spike time-dependant plasticity [102], allows for unsupervised learning where neurons learn the structure of input patterns, rather than classification labels.

Another recent development in Bio-inspired networks suggests using both a bio-inspired architecture and a bio inspired learning rule [72], where typically the architecture is the only thing replicated in bio-inspired networks. The literature shows the use of spike timing-dependant plasticity to replicate the visual cortex of a Mamalian, and shows an improvement in results which was able to outperform the DeepConvNet on the datasets tested, with an improvement of 10.2% on the 3D objects dataset when using 500 features.

2.5.4 Tactile Sensors for Invariant Object Recognition

Tactile sensors are another new area in object recognition. By using sensors attached to a robot arm a 3D model is able to be produced to provide more data for a machine learning algorithm to be able to identify [92]. Using a sensor to build up a 3D model can help with

invariance by ignoring the clutter of unrelated objects that would have an effect on an image, also allowing for the object to be viewed by the algorithm on different planes of view.

Tactile sensors would also be useful in 2D images, and could provide more data about the objects position and scale if combined with visual object recognition. Although for a majority of object recognition cases we would like to be able to identify images without having to have some interaction with the object, as physical interactions mean that expensive hardware needs to be deployed and time consuming data collection also becomes a large problem if trying to scale this solution.

While this method is interesting, it is best suited for use with machine learning applications involving robotics. For the purpose of rotation invariance alone it would be an over engineered solution to the problem at hand.

2.5.5 Rotation Invariant Galaxy Identification

The literature uses a deep learning based feed forward network for recognising categories of Galaxies [39] along with a convolutional structure. To recognise the galaxy invariant to rotation the images were cropped, flipped, scaled, rotated, and translated, with the results of each change being averaged together for the overall result, and obtained an accuracy of over 99%.

The dataset was based on and inspired by the GalaxyZoo project, which allows members of the public to help categorise the images of galaxies from multiple telescopes, including the Hubble. Some pictures of galaxies are inherently rotation invariant due to their circular nature, but due to a lack of a direction such as up or down this becomes an important area for rotation invariance.

This application in this literature is interesting as it is a perfect application for a rotation invariant machine learning method due to the lack of orientation in space, though the method of producing multiple images at different variations could be called a brute force solution to the problem.

2.5.6 Slow Feature Analysis

Slow feature analysis is a method to take a multidimensional input and find a scalar input to output function while extracting slowly varying features amongst a quickly varying input [49][138]. The idea of this method is to pass enough information about an object for it to be recognised, but to also reduce variations that occur that make images vary from each other. The architecture related to using this network is different to a standard neural

network and is similar to a convolutional neural network where the neurons are arranged in a multidimensional structure for each layer, instead of just one dimension per layer.

This is an interesting method that deals with variance well, however is not well equipt to deal with multiple objects presented at once in the same image like a convolutional neural network is able to do.

2.5.7 Neocognitrons

Neocognitron is a network that is self organised and unsupervised [51]. It is capable of recognising patterns through geometrical similarity and allows for shift invariant recognition of images. Neocognitron networks are constructed with a cascade of structures with each structure consisting of a number of layers of different types of cells. These different types of cells are based on the visual cortex system where there a multiple types of cells which are high or low hyper-complex cells. This is different to a standard neural network where all the cells or neurons are of the same type. This is an interesting method that is similar to the ideas of Bio-inspired learning by attempting to replicate the visual cortex system.

2.5.8 3D Object Recognition with Local Surface Features

3D object recognition brings computer vision and object recognition closer to human vision and recognition standards by presenting a machine learning algorithm with a 3D image through 2 images captured at the same time at a uniform distance from each other. 3D object recognition can also help identify an object among clutter and other background textures that would normally be difficult to identify, as well as help provide some data on the distance of the object from the camera.

Local surface features are created by extracting key points or features from an 3D image, where the area (space) around the key point is then used to create a feature descriptor [60]. Feature descriptors can be generated from a large number of methods.

A 3D model is an ideal input for identifying an object due to retaining data that would be lost in a 2D image. However the extra data provided by a 3D image could provide a growing exponential amount of positions that the 3D object could be presented in.

2.5.9 National Data Science Bowl 2015

A challenge presented at the National Data Science Bowl 2015 [15] had participants compete in creating machine learning methods to study the health of the oceans and identify different types of plankton. The winning team ‘Team Deep Sea’ presented their results of 82% accuracy

on the classification set, using a number of combined methods with a deep convolutional neural network [38].

The work presented used a varying number of layers up to 16 layers in total. Their results showed an increase in network performance for each layer introduced to the network. The team also made use of Cyclic pooling which created a stack of convolutional layers which in parallel, each view a rotation and/or flip of the original image. They also present the idea of rolling feature maps, where the outputs of the cyclic pooling stack can be combined as a stack for the next layer to use as an input. The use of cyclic pooling in this way gives the network some invariance to rotation and flips of an object in image recognition, and is very useful in this application due to it being likely the objects (plankton) will be at different angles and views.

This is similar to the method presented in Chapter 5 which uses parallel layers to apply filters in multiple rotations, however in this instance each parallel layer is the same but each receive a different version of the input image that has been rotated for flipped.

2.5.10 Rotation invariant convolutional filters for texture classification

The closest related works to this project is the use of rotational filters for patterns and textures [55], that was published at the same time as this project was developing the same new method. The paper introduces the novel method of introducing rotation invariance into a Convolutional Neural Network through a number of identical filters tied together but at different rotations within a convolutional layer.

This paper uses this novel method for shallow Convolutional Networks and applies the method to a dataset of patterns and textures to be able to identify them while being invariant to rotation. Filters are cropped into circular filters and duplicated with rotation. The different filter rotations are then pooled into a single output that represents a rotational invariant output for the original filter. On the textured dataset the method is able to produce results similar to state of the art methods, and improves greatly on standard convolutional neural networks.

Chapter 5 defines this method in more detail as this thesis creates an extension of this method for Deep Convolutional Networks to be used with object recognition.

2.5.11 Summary

A number of existing methods in this section have improved on the area of rotation invariance or other transformation invariance, and while some are effective, others have disadvantages that make them complicated or difficult to use practice. It is difficult to define whether many of these methods could be called state of the art, but many of them greatly improve results

on data with large variations, and are a great improvement on the standard method of brute force (Section 2.2.2.1).

The method presented in this thesis works to reduce complexity by working directly with convolutional neural networks without much need to change or alter data being inputted into the network, providing a solution where brute force of an input image is no longer required.

Chapter 3

Deep Dropout for Digits and Characters in Natural Images

This application was presented as a paper [10] at the International Conference for Neural Information Processing (ICONIP) 2015. Information on where to download the paper can be found attached at Appendix A. This chapter presents an application of dropout on deep neural networks to improve results in recognising natural characters and numbers in images from the CHARS74K dataset (detailed in Section 3.1.2).

3.1 Methodology

This section outlines the research questions of this chapter and the design for how the research will be conducted.

3.1.1 Research Questions

The following research questions have been developed as part of this chapters research application.

- Does Dropout improve accuracy results for Deep Networks used for natural character and digit recognition.
- Does an extended training time effect the learning of the dropout algorithm
- Does the network size have a large effect on the Dropout methods effectiveness.

3.1.2 Dataset

Experiments in this chapter use the CHARS74K dataset [34] which is a dataset of natural images for characters and numbers, as well as some synthesised fonts. The subset of characters and numbers from natural images is the only subset from this dataset for use with experiments.

The Chars74K dataset is a natural image dataset of characters and digits from the real world. The dataset consists of 74 thousand images in 62 classes (0-9, A-Z, a-z), with images obtained from three sources (natural images, hand drawn images, and synthesised characters), unlike MNIST which consists of only handwritten digits.

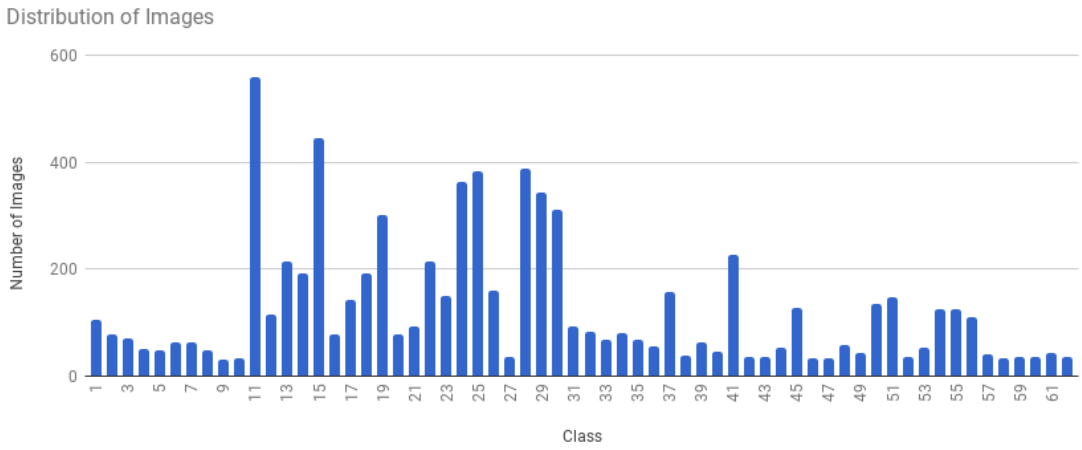
The natural images subset of this dataset consists of 7705 images, which while small, is a useful dataset for real world applications that involve automatic text recognition, such as number plate recognition, house number recognition, and other applications that do not involve written text.

The hand drawn images would be best used for written document recognition with the synthesised characters sub set best suited to recognising text in printed documents. Both of these two subsets are very useful for reading documents, but have their limitations on applications in natural images.

Fig. 3.1 A sample of images from the data set

Chapter 5 also uses the Chars74K dataset and specifically the natural images subset. Figure 3.2 shows the distribution of images in the dataset between the 62 classes. It can be noted that some of the most frequently occurring classes are vowels.

Fig. 3.2 Distribution of Chars74K images



3.1.2.1 Dataset Processing

In the dataset there are a number of images that are of different colours and sizes, which makes the learning process more complex. To make the learning process simpler, pre-processing has been applied to the images.

Images are first grey-scaled to remove colour channels from the image. For the purpose of recognising characters and digits, the extra information provided by colour would give little improvement for learning. The difficulty with grey-scaling this dataset is that some characters are colour, some are black, and some are white, which means after grey-scaling not all the characters and numbers will be of the same colour.

To combat this we can try and work out if the background is white or black using an average of the pixels bordering the image, and then flip the colours if need be. This worked in a majority of cases, but in a number of cases where the background and character was the same colour due to character outlining, the characters remained the wrong colour. This was minimal and within acceptable levels for the experiment being conducted.

The threshold method was applied to the images to further remove unnecessary data and improve the distinction between the characters/digits and their backgrounds. For a majority of images this caused no issue, but with a number of images where the character and background were too close in colour, the entire image was blacked out. As with the previous step, this was uncommon and within acceptable levels, and effected images were left in the dataset so that results would closer identify with data that was automatically processed when in use in the field.

The final change that needed to be made to this dataset was to make the images all the same size. A size of 50x50 pixels was selected and the images were scaled down along the

Fig. 3.3 A sample of CHARS74K images after processing

longest edge (either width or height), and then padded on the smallest edge with black to match the background colour.

Figure 3.3 shows a sample for the processed dataset. It can be seen that some images have been blanked out by the process (Row 1, Col 4), and that some images have had characters and backgrounds that are the same colour due to character/digit outlining (Row 2, Col 6). A majority of the images have been correctly processed and simplified from the original images that can be seen in figure 3.1 in Section 3.1.2. Pre-processing the dataset in this way helps to induce invariance into the network as it removes variations in lighting.

3.2 Methods

Dropout (Section 2.4.4.5) and Deep learning (Section 2.4.2.2) are the main methods in this application, applied with a Restricted Boltzmann Machine (Section 2.4.2.1) to pre-initialise the weights of each network. These methods are combined to conduct experiments on the natural image dataset, with dropout being used on all layers (except output) with the same drop ratio for each hidden layer of 30% and a drop rate of 20% for the input layer.

The reduced drop rate for the hidden layer can ensure that there is not too much loss of useful information from the input images going into the network. Previous research has found a drop rate of up to 50% to be effective in removing overfitting, but for this application no more than 30% was selected.

3.3 Experiments

A number of experiments will be conducted in this chapter. As previously described a number of different architectures will be tested so that differences in dropout effectiveness between

architectures can be observed, if any differences are to be found. Different architectures are also tested due to the difficulty in finding the correct hyperparameters for neural networks without using trail and error (Section 2.4.3).

The below table outlines the 10 architectures that have been selected for experimentation. Some architectures have 3 hidden layers while others have up to 5 hidden layers, and a variety of different layer sizes have been selected between the experiments.

Table 3.1 Architecture Definition table for Dropout Application Experiments

Architecture ID	Architecture
Architecture A	[1000,500,500]
Architecture B	[1500,1000,500]
Architecture C	[1500,1000,500,250]
Architecture D	[500,500,500]
Architecture E	[500,500,1000]
Architecture F	[1500,1000,500,1000]
Architecture G	[500,500,200]
Architecture H	[800,500,500,200]
Architecture I	[500,500,500,500]
Architecture J	[800,500,500,500,250]

10 Restricted Boltzmann Machines were initialised with the same architectures defined in the table above to act as pre-loaded weights for each architecture test. Each layer (1 RBM on the Stack) was given a total of 10,000 iterations to pre-train weights. Both the dropout and non dropout benchmark for each architecture takes its weights from the same RBM stack.

3.4 Results

Table 3.2 shows the results of the 20 experiments (10 architectures x 2 methods) after 10,000 iterations of training with and without dropout. Out of the 10 experiments conducted it was found that dropout improved results of 8 out of 10 of the architectures, with 2 of the dropout architectures performing worse than their benchmark counterpart. Over the 10 experiments an average improvement was found of 1.52% which increases to 1.9875% when discounting the architectures that dropout was not effective on.

The best dropout network across all of the experiments achieved an error of 42.9% while the best non-dropout network error was 44.6% which happened to be the same architecture, giving an increase of 1.7% between each methods best results. We can see that Architecture D (500,500,500) and Architecture F (1500,1000,500,1000) were the two networks where

Table 3.2 Test Results (10,000 iterations) for Dropout Application Experiments

Architecture	Error With Dropout	Error Without Dropout	Dropout Improvement
Architecture A	45.7	46.9	1.2
Architecture B	46.4	47.3	0.9
Architecture C	46.6	49.1	2.5
Architecture D	45.4	45.3	-0.1
Architecture E	43.7	45.8	2.1
Architecture F	46.6	46.0	-0.6
Architecture G	45.7	48.3	2.6
Architecture H	45.4	47.6	2.2
Architecture I	42.9	44.6	1.7
Architecture J	44.1	46.8	2.7

dropout was worse in predicting the test data. It is difficult to identify the reason these architectures were worse because while Architecture F has the largest number of connections by far, Architecture D is not the smallest network in terms of connections which would be Architecture G which achieved an improvement of 2.6%. It is also interesting to note that the larger improvements were seen on networks with a smaller last hidden layer.

The networks were allowed to continue training until 15,000 iterations as typically dropout sees an improvement and need to be run for more iterations than a traditional deep network [75]. Table 3.3 shows the results of the extra 5,000 iterations. We can see that a number of the networks had either improved or not changed where the network may have already converged by 10,000 iterations. In one case the error on one dropout network went up by 0.1% which is negligible and within the network jumping in and out of local minima. A number of traditional networks error results got worse with the extra iterations suggesting they started to over fit the solution.

Of the two networks that had previously done worse, dropout either overtook the benchmark network during the extra training or closed the gap to 0.1% between itself and the benchmark, and closed the gap entirely between itself and the benchmark at 10,000 iterations. This suggests that an increased training time is both necessary and beneficial for dropout networks. The smallest dropout error dropped to 42.7% on Architecture I and the average accuracy increase rose to 1.75%. There is little evidence to suggest the size (large or small) of the network had a major effect on the effectiveness of dropout over the non-dropout benchmark, suggesting that finding the best architecture in general through usual trial and error techniques is best.

Figure 3.4 shows the loss of the validation set over 10,000 iterations for one of the networks. The left hand graph shows the loss for dropout in blue, and the loss of our

Table 3.3 Test Results (15,000 iterations) for Dropout Application Experiments

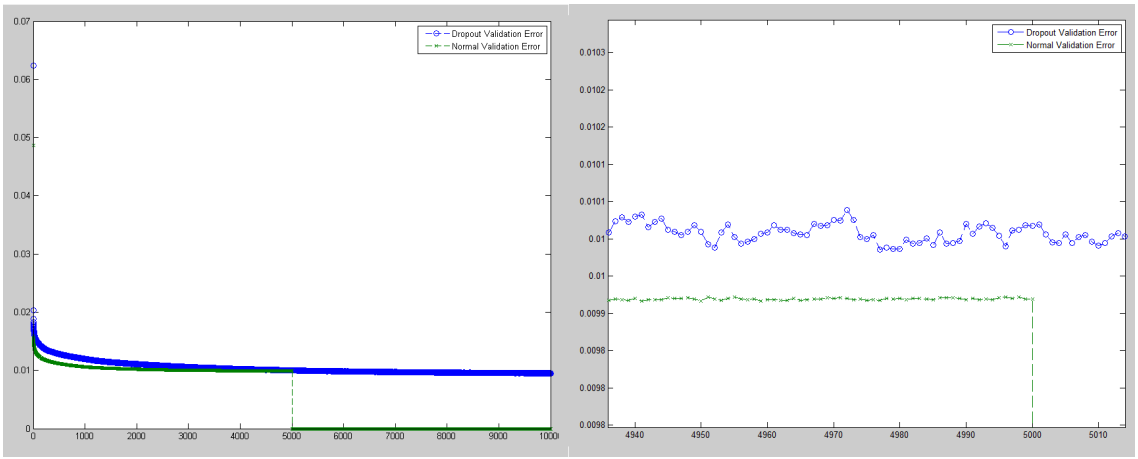
Architecture	Error With Dropout	Error Without Dropout	Dropout Improvement
Architecture A	45.7	46.7	1.0
Architecture B	46.4	47.5	1.1
Architecture C	46.4	49.1	2.7
Architecture D	44.4	45.3	0.9
Architecture E	43.8	45.6	2.1
Architecture F	46.0	45.9	-0.1
Architecture G	45.7	48.3	2.6
Architecture H	44.8	48.1	3.3
Architecture I	42.7	44.7	2.0
Architecture J	44.1	46.8	2.7

benchmark in green. The right hand graph shows a zoomed version of the first graph, looking at the point where early stopping caused the benchmark to stop learning.

The benchmark network converged at around 5,000 iterations and at this point in time had a better loss than that of the dropout network, however over the remaining time we can see that the dropout network reaches the same point and starts to overtake the loss of the benchmark over more iteration, further showing the need for dropout to have longer training runs.

In the right hand graph we can see that the benchmark network has a very smooth loss line, suggesting that it is stuck in a local minima, but the dropout loss line is very jagged and jumps up and down in results, suggesting that dropout is causing it to jump out of local minima between some iterations to explore another minima.

Fig. 3.4 Validation Error During Training for the [500,500,500,500] hidden layer architecture.



3.5 Conclusion

It can be concluded that using the dropout method for deep neural networks is an effective method for improving accuracy of image recognition challenges for characters and digits in natural images. Over a total of 15,000 iterations it is found that an average improvement of 1.75% was made by the dropout method, with an accuracy of 57.3% (Error of 42.7%) on the best architecture tested. These results are in line with [34], which using Multiple Kernel Learning Experiments achieved an accuracy of 55.26%, though while using the same image subset the classes were balanced to 15 test and 15 training examples per class, compared to the random sample selection used in this chapters experiments.

It can also be concluded that dropout networks should be given an extended training time for the method to learn, as due to a percentage of neurons switched off for training, this naturally means it will take longer to train, and combined with dropouts ability to avoid getting stuck in a local minima adding further to this training time. It is possible that some of the results here could be improved further with further iterations while some other networks had already converged. There is also the possibility that further fine tuning of network architecture could yield improvements for both dropout and a traditional deep network without dropout.

Chapter 4 goes on to further the dropout method and improve results yielded from networks employing the method.

Chapter 4

Selective Dropout

The methods presented in this Chapter were presented as a paper [9] at the International Conference for Neural Information Processing (ICONIP) in Japan 2016. Information on where to download the paper can be found attached at Appendix B. This method was originally created to be used with invariant methods presented in Chapter 5, however it was replaced with Batch Normalisation as a newer and more effective method.

Dropout has already been proven to be an effective method for reducing overfitting of a neural network [128]. This chapter explores the effect of using various selection criteria to identify which neurons would be the best neurons to switch off for a training iteration compared to the randomness of traditional dropout in an effort to improve dropouts effectiveness. The methods explained in this chapter will be tested on the well known MNIST dataset (detailed in Section 4.1.2), which is a handwritten digits dataset.

Similar work to modify dropout has been made by Ba and Frey [7] where a binary belief network is used to identify the optimal drop probability for each neuron in a neural network. This Binary belief network runs separate to the main classification network outputting the probability for neurons. Other work has been done by Duyckm et al [41] who optimise dropout using optimisation methods such as simulated annealing, and work by Wan et al [135] who introduce a method called DropConnect which works on individual weights rather than a collection of weights (neurons) as a whole.

4.1 Methodology

This section outlines the research questions of this chapter and the design for how the research will be conducted.

4.1.1 Research Questions

The following research questions have been developed as part of this chapters research.

- Can Dropout be improved by carefully selecting which neurons to switch off
- Which is the best way to choose which neurons should be switched off

4.1.2 Dataset

MNIST is a subset of the NIST dataset containing handwritten digits from 0 to 9 [83]. These digits have been normalised to the same size and padded to make a new image size of 28 by 28 pixels, giving 784 pixels in total.

The full MNIST dataset contains 70,000 images, consisting of two defined subsets for training (60,000) and testing (10,000). The figure below (4.1) shows a small random sample of the MNIST dataset.

Fig. 4.1 A Sample of Images from the MNIST Data set

MNIST is a particularly popular dataset for image recognition applications due to the large size of the dataset, and the fact that all images are uniformly sized and centred. The dataset is an ideal one to use for testing out new methods without interference from the typical variance found in images. Due to the wide usage of this dataset it is also a great choice for benchmarking against results presented in other papers, and the size and high accuracy usually obtained in classifying the dataset allow for easy comparisons and evaluations on how well a new method might be performing.

MNIST has been selected for this experiment due to its wide use in the field and its uniform nature which will help to remove other factors from skewing results.

As discussed 10-fold validation will be used for these experiments. The dataset is split into 10 sections with one section of 6000 used for testing and the remaining 54,000 images used for training and validation in and 90% and 10% spilt respectively. No extra processing is performed on the base MNIST dataset which has already been performed by dataset creators.

4.2 Methods

4.2.1 Traditional

The Traditional or uniform dropout method as described in Section 2.4.4.5 is a method used to reduce overfitting in neural networks. During training a random selection of neurons are switched off from dropout layers, and are then switched back on and scaled for use and testing. In the results table traditional dropout is represented by **TD**.

4.2.2 Selective Dropout

Selective Dropout however differs from traditional dropout by attempting to intelligently choose which neurons should be switched off on each iteration rather than randomly selecting them. This method hypothesises that increasing or decreasing the probability of a neuron to be dropped in proportion to a given statistic could give an improvement over traditional methods. Three methods are investigated in this chapter, which can be described as:

- Looking at the average change in weights between training iterations
- Looking at the neurons average weight size
- Looking at the variance of a neurons output during a training iteration.

Each method creates a statistic which can be used to alter neuron drop probabilities. Each layers total drop probability remains the same as it would be in traditional dropout (selected by the user), however each individual neurons probability is scaled and changed relative to one another so that the average probability of all the neurons equals that of the layer drop probability as a whole.

Equation 4.1 comes from the need for the probability of a neuron to be proportional to a positive statistic calculated so that the higher a statistic is the higher the probability that a neuron will be chosen to be dropped. Combined with equation 4.2 the probabilities calculated are not only proportional to the statistic but also meets the need that the average of all probabilities matches that of the user given drop percentage for the whole layer.

$$\mathbf{P} = C\mathbf{S}^{\alpha} \tag{4.1}$$

P in Equation 4.1 represents a vector of probability for all the neurons in a particular layer. **S** represents the statistical values that are calculated for each neuron that cause the scaling of

the neurons probabilities. These statistical values are defined in the following sections. C represents a constant value that allows the probabilities to be calculated proportionate to the statistics and the given layer drop rate. Equation 4.2 shows how the constant C is calculated. Without this value then we would not be able to keep the average of all the calculated neurons equal to that of the layers drop rate (d). In Equation 4.2 N is used to represent the expected number of neurons to be dropped from a layer, such that $N = C \sum \mathbf{S}^{\alpha}$. How N is calculated can be shown in equation 4.3 with d_k being the neuron drop rate for layer k.

$$C = \frac{N}{\sum \mathbf{S}^{\alpha}} \tag{4.2}$$

α in equation 4.2 is a constant between 0 and 1 chosen at numerical decrements starting at 1 until $Cmax(\mathbf{S}^{\alpha}) \leq 1$ is satisfied. This ensures the calculated probability is always less or equal to 1 which would otherwise not be a probability. It also allows a wide range of statistical values to be used no matter their size. An α value of 0 in this instance would also cause traditional dropout to be run.

$$N = size(\mathbf{S}) \times d_k \tag{4.3}$$

Equation 4.1 gives a high drop probability to high statistical values, and could be considered dropping neurons in a descending order (from high to low). Some experiments also need the ability to lower the probability of a high statistic neuron from being dropped. This is accomplished using equation 4.4 where data is essentially flipped. The value 1.1 is used to ensure all statistical values are above zero so that they still have a slight probability of being switched off.

$$\mathbf{S_{new}} = 1.1 \times max(\mathbf{S_{old}}) - \mathbf{S_{old}} \tag{4.4}$$

The methods described in the following sections can be applied to neurons on any hidden layer, however can not be applied to the input layer as the input layer has no incoming weights or the output of the layer is a set image. In this instance traditional dropout is used on the input layer with random dropping, and the methods being tested are used on the remaining hidden layers.

4.2.2.1 Weight Change

The first method to be tested will be Average Weight Change. Average Weight Change looks at the differences in the weights for a neuron before and after a training iteration and takes an average of the result. This should allow for a statistic that will show how much a neuron has learnt during a training iteration and by how much a neuron has learnt.

The equation below (Equation 4.5) shows the method of finding the average weight change for each neuron where avg_k is the nodes input weight change average at node k, and $w_{jk}^{(t)}$ is the j^{th} weight for node k at the beginning of iteration t (Before training). n represents the number of weights feeding into a neuron from the nodes in the previous layer.

$$\mathbf{avg}_k = \frac{1}{n}\sum_{j=1}^{n}(|w_{jk}^{(t)} - w_{jk}^{(t-1)}|) \tag{4.5}$$

With neurons switched off during a training iteration we come across an issue where the average change in weights for that neuron would be returned as 0, as it wouldn't have had a chance to train during the iteration. This is a problem that can be seen in some of the other methods that follow this one, meaning we do not have a statistic to be generated for our probability calculations, and as such, a low statistic could cause neurons to flicker on and off or stay off entirely. A solution to this problem is to save the average weight change from the previous iteration for each neuron before they are switched off, and then use this result for comparison with the updated statistics with neurons that were switched on for the training iteration.

With the average change in weights calculated by equation 4.5 it is then possible to work out which neurons have a high or low change in weights between iterations where equation 4.1 can then be used to assign drop probabilities to each of those neurons, and to allow a larger drop probability to be given to neurons with either a big or small change in weights.

Giving a higher probability to neurons with the smallest average change in weights allows us to increase their probability of being switched off, and is much like sorting our statistic in Ascending order and giving high probabilities to the first x neurons in the vector. This could be considered as neurons that have finished learning as their weights are changing the least. In the results table the results of average weight change dropout (dropping smallest average change) are represented by **WA**.

Giving a higher probability to neurons with a high average change in weights, will allow us to increase the chances of neurons with a high change in weights being switched off. This could be considered as increasing the chances of switching off the neurons that are learning,

in order to force those that are not learning to learn. In the results table the results of average weight change dropout (dropping highest average change) are represented by **WD**.

4.2.2.2 Average Weight

The second method to be considered for improving the dropout method is Average Weight. Average weight looks at an average of all the weights for a particular neuron to be used as a statistic. The motivation for this is that neurons are initialised as very small values and are typically very close to zero, meaning that neurons with a high set of weights may have learnt more than neurons with low average weights.

Equation 4.6 shows the calculation to find the average weights for neurons in any given layer. In this equation avg_k represents the average input weight for node k and ${w_{jk}}^{(t)}$ is the j^{th} weight of node k during iteration t before training has taken place.

$$avg_k = \frac{1}{n}\sum_{j=1}^{n}(|w_{jk}^{(t)}|) \tag{4.6}$$

In the previous method we calculated a base statistical representation for each neuron to allow for a comparison value for neurons switched off. With this method there is no before and after comparison between weights and the standard weight matrix is used to calculate the statistic, and as such because weights are automatically replaced for dropped neurons after training there is no need to keep a representation of switched off neurons statistical values.

By giving the neurons with high average weight values higher drop probabilities we can increase the chances of switching off neurons with high weights attached to them. The idea of this method is that weights are initialised close to zero and that weights that are closer zero will have learnt less and should be kept on to learn while the neurons that have learnt are switched off. Another motivation for this method is regularisation, typical regularisation methods penalise large weights in some way, and this method also does so by preventing neurons with very large weights from increasing. In Section 4.4 the results of average weight dropout (dropping highest average weight) are represented by **AD**.

On the other hand giving a high drop probability to neurons with low average weights allows us to increase the chances of switching off neurons where the average weight is of a small value. It is expected that this could cause neurons with small initial weights to be given a high drop probability for a majority of the training, causing them to spend a majority of training time switched off and rarely being switched back on, in turn leaving the same neurons learning a majority of the time. This experiment can still be run to give some comparison of the effect between giving high probabilities to switching of neurons with

high or low weight magnitudes. In the results table the results of average weight dropout (dropping lowest average weight) are represented by **AA**.

4.2.2.3 Output Variance

The final method presented for testing in this section is Output Variance. Output variance is a measure of the variance in the output of a neuron over the period of an iteration. The motivation for this is that it may be possible to tell if a neuron has learnt anything useful by the amount of variation in output between images. A neuron that has a low variance and always outputs the same value with a changing input may be considered as not having learnt any useful information to help distinguish between classification classes.

Equation 4.7 shows how the statistic for this method is calculated by using the output matrix for the previous completed iteration. $N\_Variance_k$ in the equation represents the output variance for the neuron k and $\mathbf{X}_k^{(t-1)}$ is the output vector for node k for iteration $t-1$ (the previous iteration). This variance is calculated in the dimension that holds neurons outputs for a given iteration or batch.

$$N\_Variance_k = variance(\mathbf{X}_k^{(t-1)}) \tag{4.7}$$

Much like the Average Weight change method, the Output Variance method requires a baseline for each switched off neuron as switched off neurons would provide a variance of 0. The previous iterations output variance is saved for each switched off neuron, but in the case of the first iteration where there is no previous iteration to get data from we run the network with dropout switched off to first generate a base statistic for each neuron. Once the statistic has been generated it can then be used with equation 4.1 to give high probabilities to neurons with either high or low output variance.

Giving a high probability to neurons with high output variance will allow us to increase the chance of switching off neurons with high variance. High variance could indicate that the neuron has learnt an interesting feature and no longer needs training and as such can be switched off to allow neurons with a lower variance to learn. In the results table the results of neuron output variance (dropping highest variance) are represented by **VD**.

Giving a high probability to neurons with a small output variance allows us to increase the chance of switching off neurons with low variance. Low variance may indicate that a neuron is outputting roughly the same value no matter the input and has learnt no interesting features. It is expected that switching off these neurons that have not learnt, will cause the network to learn at a very slow rate and possibly be less efficient than the traditional dropout

method. In the results table the results of neuron output variance (dropping lowest variance) are represented by **VA**.

4.3 Experiments

Experiments will be conducted over approximately 10,000 iterations which should be plenty for the MNIST dataset. Each method is tested in two ways (High and Low probability for high statistics) giving us 6 experiments to test the 3 methods, as well as a 7th experiment to create our benchmark with traditional dropout. Each of these experiments will be tested on two network architectures creating 14 experiments, and then run in 10-fold validation to create a total of 140 experiments.

Two stacks of Restricted Boltzmann Machines were created, one for each architecture, in order to initialise the weights for each experiment. All experiment on an architecture will use the same initialisation weights from these Restricted Boltzmann Machines.

The two hidden layer architectures chosen are as follows:

- 500,500,500
- 500,500,1000

The above architectures were chosen as they performed well with the similar CHARS74k dataset in the previous chapter, without being too large, which would cause network run-times to increase. 2 architectures were selected to allow the impact of number of neurons to be observed.

All the networks will use the same dropout rate of 20% on the input layer and a dropout rate of 40% on all the hidden layers in the network. The output layer as previously discussed has a drop rate of 0% as we do not want to drop neurons from this layer.

4.4 Results

Table 4.1 shows the results of experiments conducted for each method and architecture. Error rates are the product of averaging the result from each of the 10-fold networks for that instance.

It can seen from the table that all of the three methods produced results better than traditional dropout methods by giving higher drop probability to smaller values of the calculated statistics. Weight change in ascending order (**WA**) gave a result 0.378% better on average, average weight in ascending (**AA**) order gave an improvement of 0.414% on

Table 4.1 Selective Dropout Experiment results

Method	% Error [500,500,500]	% Error [500,500,1000]
TD - Traditional Dropout	4.7300	4.5933
WD - Weight Change (Descending)	7.0533	7.5783
WA - Weight Change (Ascending)	**4.3167**	**4.2517**
AD - Average Weight (Descending)	5.2983	5.6533
AA - Average Weight (Ascending)	**4.3217**	**4.1733**
VD - Output Variance (Descending)	14.4633	13.7833
VA - Output Variance (Ascending)	**3.5167**	**3.4667**

average, and output variance in ascending order (**VA**) gave an improvement of 1.17% on average above that of traditional dropout. Both tested architectures saw improved results over the benchmark network.

Several of these results were unexpected and produced results that were the opposite of what had been predicted in Section 4.2.2. The Average Weight change method was one method that did yielded results as expected, by giving a high probability of switching off neurons to neurons with little change in weights between iterations, suggesting that the neurons were either not learning, or that they had finished learning. Giving neurons with a high weight change a higher drop probability caused reduced performance and caused it to be 2.744% worse on average than the benchmark for traditional dropout.

It was expected that with the Average Weights method that giving a higher drop probability to neurons with a higher average weight would improve the results on the assumption that high average weights would have learnt more and could be switched off to force learning of other neurons. However this gave a result that was 0.814% worse than traditional dropout on average, and instead switching off neurons with a smaller average weight during training had a much better result than this, suggesting that giving neurons with larger weights smaller drop probabilities was a much better method. This could be perceived as leaving neurons that have learnt more or are still learning to remain switched on.

The final method of output variance was also expected to perform well by giving a high drop probability to neurons with a high output variance, with the idea that neurons with a high variance had learnt an interesting feature and could be switched of to cause other neurons to learn. This however gave the worst results of all the experiments conducted, being an average of 9.462% worse off than using the traditional dropout method. Performing the opposite and giving a high drop probability to neurons with a small output variance produced the best performing network with an improvement of 1.17% over traditional dropout on average.

Possible explanations for the last two methods that performed as unexpected could be that what could be considered as the worst neurons were in-fact hindering the networks performance and that it would be better to switch them off during training. Another explanation could be the possibility that the dropout was causing a temporary form of pruning [69] or sparsing [88], causing the network to prune or sparse part of itself for a majority of the training period, which could be consistent with what can be seen in biology.

4.5 Conclusion

It can be concluded that the traditional method of Dropout can be improved by selectively choosing which neurons to drop during training iterations by probabilistically choosing neurons with smaller statistical values in the methods described (Equations 4.5, 4.6, and 4.7).

From the three methods tested it was found that the best method for improving dropout was giving a high probability of being dropped to neurons with a low output variance between iterations. An average improvement of 1.17% was found using this method. Dropping neurons with the smallest change in weights between iterations and a smaller weight size average gave small improvements of 0.378% and 0.414% on average.

It can also be concluded that a bad method of selecting neurons to be dropped can give results worse than that of traditional random dropout. In the case of giving a higher drop probability to neurons with a high output variance caused the accuracy results of that network to be an average of 9.46% worse than the average for traditional random dropout. This solidifies that selecting the correct neurons for dropout can have a massive difference on experiment results.

Chapter 5

Rotation Invariant Convolutional Neural Networks using Parallel Rotated Filters

The main focus of this thesis is Invariance, and while previous chapters have looked at applications of deep learning and new methods to improve deep learning they have not specifically focused on invariance.

This Chapter looks at creating a rotation invariant machine learning method using convolutional neural networks. As previously discussed in Section 2.5.10 this method was published in "Learning rotation invariant convolutional filters for texture classification" by Gonzalez et al [55], at the same time as this thesis was introducing the same method. This thesis however also takes the described method further with an application to deep convolutional neural networks, and is provided with more complicated image datasets which are applicable to more real world applications than that of simple pattern or texture recognition. This thesis also introduces a version of the open source Caffe framework implementing this method into the existing CNN code architecture.

The method works by taking a traditional convolutional neural network and adapting it for rotation invariance. This involves combining circular filters in convolution layers as well as running a number of convolutional layers in parallel, each with filters tied to the other parallel layers but at a rotation of the original. This is discussed in detail in Section 5.1.

A network with rotational invariance can be a useful feature for real world applications where a lot of rotation is found in images. These transformations can be caused by camera angles as well as objects being placed on uneven ground. Images taken by humans also inherently give a slight rotation due to camera shake, which may not be present in commercial training sets that have been produced on specialist equipment and using cameras on tripods. Applications can also follow to face recognition where a human face may not always be presented in a directly upright fashion, such as a subject laying down, or a subject tilting

or leaning their heads on another object. Combined with Convolutional Neural networks this method also becomes invariant to small shifts of the object in an image due to the way convolutional networks apply filters to images. Other applications may also be found in environments where there is no up or down orientation in an image, such as space where there is little or no gravity to give an indication that a particular orientation is dominant. This is prevalent in work identifying images of galaxies or satellite images looking down on earth from above. Aerial images also have this same issue as they look down on an subject with no particular orientation.

5.1 Methods

The base of this method can be split into multiple steps. The first step performed during initialisation and reinforced during training, is the conversion of a square filter into a circular filter to allow for easy rotation with little data loss. The second step performed at the beginning of forward propagation is creating duplicate convolutional layers with tied weights, each at a different rotation. The third step performed during forward propagation is to then combine the result of each parallel layer in a winner takes all fashion for each node. In backwards propagation the deltas are split between the parallel networks according to which nodes won during forward propagation, and after backward propagation weights updates are then de-rotated and combined back into the original non-rotated filter layer before the process is repeated.

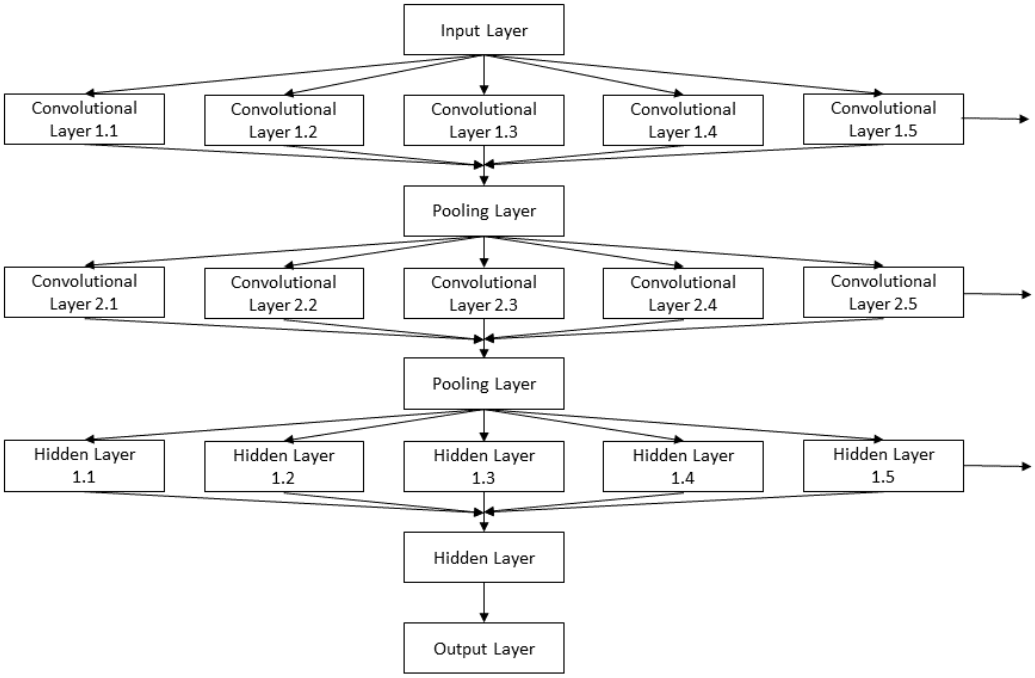
The number of parallel layers/filters depends on the rotation increment selected. A smaller increment value means that more parallel layers will be run which will cause the run-time of experiments to increase, but theoretically would be able to deal with more rotations in an image, while a large increment would mean less parallel layers, but theoretically would mean the network would not be able to deal with as many rotations. If a network took an increment of 90 degrees then 4 parallel layers would be initialled at 4 different rotation orientations (0, 90, 180, 270), with 0 degrees and 360 degrees being the same rotation. A 15 degree increment would mean 24 parallel layers in 24 different orientations (0, 15, 30, 45, ...). 360 should be directly divisible by the selected degrees of increment with no remainder to ensure equal coverage across 360 degrees without leaving a small or large gap in rotation between 0 degrees and the last parallel layer.

By applying this method to the convolutional layers in the network, the network becomes co-variant to rotations, and in order to obtain full invariance to rotation the input into the first fully connected layer of the traditional deep neural network portion of the network will need to be presented in the same number of rotations as the convolution layers. During

implementation this can easily be done by creating another convolutional layer with parallel rotated layers to masquerade as a fully connected layer where the filter size is the same as the input.

A typical network with 2 convolutional layers and pooling combined with 2 hidden layers could look like the network presented in figure 5.1, where the convolutional layers and the first hidden layer are transformed into a number of rotated parallel layers with tied weights.

Fig. 5.1 A Parallel Convolutional Network



5.1.1 Circular Filters

For this method circular filters are used, this is not a new concept as filters can take different shapes to the standard square [109], but would be considered uncommon. With traditional square filters the process of rotating filters would cause loss of data in the corners of the filters due to cropping of the filter, unless the filter is rotated only in 90 degree increments. Circular filters remove this data loss from rotation and allow the rotated filters to fit in exactly the same space without any need for cropping.

To produce circular filters a traditional square filter is used and tuned into a circular filter by changing weights outside the circular structure to zero. To decide if a weight needs to be set to zero we calculate the weights distance from the midpoint of the filter and check it against a threshold for the filter. If the weight is over the threshold then it is set to zero.

To calculate the threshold we take the size of the smallest dimension of the filter (either width or height) and apply equation 5.1 where T is our calculated threshold and S represents the size of the smallest dimension. The smallest dimension is used to ensure the circular filter has a even height and width in the event of the network being initialised with rectangular filters. -0.7 is used in the equation to allow for more data to be represented where the edge of a circular filter is close or touching the square filter edge that bounds the circular filter.

$$T = (S - 0.7)/2 \tag{5.1}$$

The distance of a weight from the mid point is calculated with equation 5.2. X and Y are the co-ordinates of the weight in the filter weight matrix, we can see 0.5 is added to these in the equation as to find the mid-point of the location as if each weight represented a square in a grid. X_{Mid} represents the midpoint of the width of the filter, and Y_{Mid} represents the midpoint of the height of the filter. D represents the calculated distance for a weight from the middle of a filter.

$$D = \sqrt{((X - 0.5) - X_{Mid})^2 + ((Y - 0.5) - Y_{Mid})^2} \tag{5.2}$$

Figure 5.2 shows a representation of a filter that was initialised as a square and was then cropped into a circle ready for the method of this chapter to be implemented. The filter shown in the figure is presented as the number eight for easier understanding as usually filters are a lot smaller and less detailed.

Fig. 5.2 Square to Circular filter conversion

As filters are cropped into circles they may benefit from using slightly larger filter sizes to their square counterparts due to the loss of the number of weights inside the filter. It is

also much easier to rotate larger filters when the rotation increment is small in comparison to rotating a small filter with a small rotation increment which would be hardly noticeable.

5.1.2 Filter Rotation

Before forwards propagation can proceed filters need to be copied from the base convolutional layer which is the 0 degree master layer, across to all the parallel layers while at the same time being rotated.

Figure 5.3 shows an example of an un-rotated filter and what the rotations of that filter would look like when rotated into their respective parallel layers. As with the previous figure, this filter is presented as the number eight for easier understanding of the process.

Fig. 5.3 Example rotations of a filter

To rotate a weight to a new position in the filter matrix its current position is taken and applied to equation 5.3 and 5.4 to find the new set of coordinates for each dimension. In this equation R represents the degree of rotation as radians. X and Y represent the new location for the weight, while X_{Prev} and Y_{Prev} represent the previous location. X_{Mid} represents the midpoint of the X dimension (width) of the filter, and Y_{Mid} represents the midpoint of the Y dimension (height) of the filter.

$$X = \cos(R) * (X_{Prev} - X_{Mid}) - \sin(R) * (Y_{Prev} - Y_{Mid}) + X_{Mid} \tag{5.3}$$

$$Y = \sin(R) * (X_{Prev} - X_{Mid}) + \cos(R) * (Y_{Prev} - Y_{Mid}) + Y_{Mid} \tag{5.4}$$

The new location that is found for a weight may fall between coordinates in the matrix. To combat this the weight is split into the 4 coordinates around it based on the distance from each point. If for example the new X coordinate does not sit between two horizontal points, then the weight value is split between the two Y dimension coordinates based on the new positions distance from the two vertical points. If the new position was exactly half way then the weight would be split equally with half of its value going to the two points. If the new point was closer to a certain point it will receive a greater part of the weight than the point furthest from it. If the new coordinates happened to be directly in the middle of 4 coordinates, then each surrounding coordinate would receive a quarter of the weight value.

As many weights are rotated they may clash into similar surrounding coordinates. Because of this when the weight is split between the 4 surrounding points it is added to a new weight matrix along with any value that may already be in the position from rotating other weights in the filter. This is a similar concept to bi-linear interpolation.

Equation 5.5 shows the first part of the method to split the weight between surrounding coordinates if the new location happens to sit between points. $left$ and $right$ represent the x axis coordinate of surrounding points, while up and $down$ represent the y axis coordinates of the surrounding points. The equation shows each direction calculates a weighting based on how far the new coordinates are from the surrounding points. lw represents the weight given to points to the left of the new coordinate location, and rw, uw, and dw represent the weight towards the right, up, and down respectively. y_{new} and x_{new} represent the new coordinates calculated in Equations 5.3 and 5.4.

$$\begin{aligned} lw &= |right - x_{new}|^2 / (|right - x_{new}|^2 + |x_{new} - left|^2) \\ rw &= |x_{new} - left|^2 / (|right - x_{new}|^2 + |x_{new} - left|^2) \\ uw &= |y_{new} - down|^2 / (|up - y_{new}|^2 + |y_{new} - down|^2) \\ dw &= |up - y_{new}|^2 / (|up - y_{new}|^2 + |y_{new} - down|^2) \end{aligned} \tag{5.5}$$

Equation 5.6 shows the second part of this method for splitting a weight between points. ul, dl, ur, and dr represent the surrounding coordinates in the top left, bottom left, top right, and bottom right respectively. The weight being split between the coordinates is multiplied

by the weighting calculated in Equation 5.5 and added to any existing weight currently in the coordinates position in the new weight matrix.

$$\begin{aligned} ul_{new} &= ul_{current} + uw * lw * weight \\ dl_{new} &= dl_{current} + dw * lw * weight \\ ur_{new} &= ur_{current} + uw * rw * weight \\ dr_{new} &= dr_{current} + dw * rw * weight \end{aligned} \tag{5.6}$$

5.1.3 Forward Propagation and Winner takes all

After the filters are rotated and split across all the parallel layers the network then needs to run forward propagation. There is no change to the actual forward propagation algorithm of the convolutional layer, however after forward propagation of each parallel layer the results need to be combined in a winner takes all fashion.

Each layer outputs a matrix where the second dimension (number of channels) is equal to that of the number of filters from the weight matrix. Each position of these matrices are compared with their counterparts and the largest absolute value is then placed into a merged matrix. For each position in the merged output matrix the winning parallel network is saved to enable back propagation to the correct winning weights.

Fig. 5.4 Example pooling of layer outputs between parallel layers

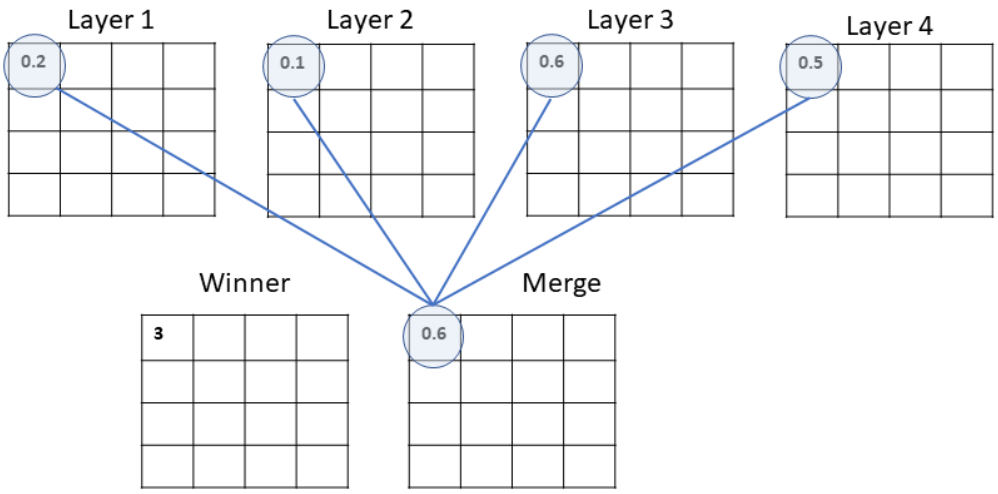


Figure 5.4 shows an example of the winner takes all on the output matrix of a single filter across a number of parallel layers. We can see in this example that layer 3 won for that particular position in the filter output and had its output moved into the merged output matrix,

as well as having it marked as the winner in a separate matrix keeping track of winners. If this step is not performed then it would not be possible to update the correct weight in the correct filter rotation during back propagation. After these steps are performed the merged output matrix is passed onto the next layer in the network as if it was the output from 1 single layer rather that a set of parallel layers.

5.1.4 Backwards Propagation

During the backwards propagation of the layer there are a number of steps that need completing before and after the back propagation algorithm can be executed. Before the backwards propagation can happen the deltas that have been passed up from the previous layer in the back propagation process need to be duplicated for each parallel network. This is not as simple as a direct duplication for each network as only the winning weight from across the parallel layers should receive the delta passed back for its position in the filter. Layers that did not win for a particular output position in the output matrix will instead present a zero as the diff so that it does not contribute to the weight updates.

Fig. 5.5 Example splitting deltas between parallel layers

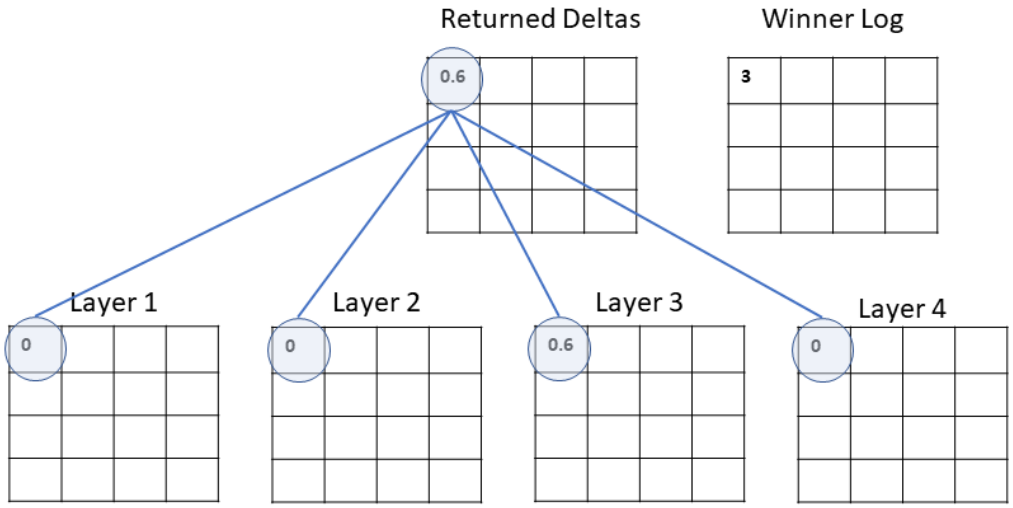


Figure 5.5 shows an example of a delta matrix returned for the set of parallel layers on a singular filter. Using the log of which filter weight won the merging process after forward propagation the matrix is split into separate delta matrix for each parallel layer. We can see that the winning layer got the delta for that position, and the remaining layers received a delta of 0 in that particular position.

After the deltas have been split the back propagation algorithm can be implemented for each parallel layer. There is no change to the usual backwards propagation algorithm employed for convolutional neural networks in this step.

After backwards propagation has been completed the deltas that have come out of the backwards propagation for the next layer are merged through simple addition. The weight updates that are calculated but not yet applied at this step then need to be merged before the weights on the master layer (0 degrees layer) can be updated.

To perform this the weight updates calculated are de-rotated for each layer based on each layers rotation. The same method as discussed in Section 5.1.2 is used to perform this action using a degree of rotation that is equal to 360 minus the original degree of rotation. Once all weight update matrices are back in their original orientation they can be merged and updated in line with normal weight updates.

5.1.5 Batch Normalisation

After a convolutional layer with parallel layers of tied filters has been implemented there is the need to regularise the data. This is because the outputs from our combined network are too large to process further through the network and continue learning.

One solution to this is to use batch normalisation after each convolutional layer to both normalise and regularise the output of the network. Batch normalisation was discussed in Section 2.4.4.6, and this method removes the need for dropout to be used with our layers. The method also reduces the number of iterations that are needed to reach a convergence point which is a large advantage with this method due to the increased run-time for running extra iterations as discussed in Section 5.5.3.

5.2 Methodology

This section outlines experiment methods and tables for testing this chapters methods. Each experiment will be provided with an experiment ID as well as each dataset to allow for easy referencing between tables.

5.2.1 Research Questions

The following research questions have been developed as part of this chapters research.

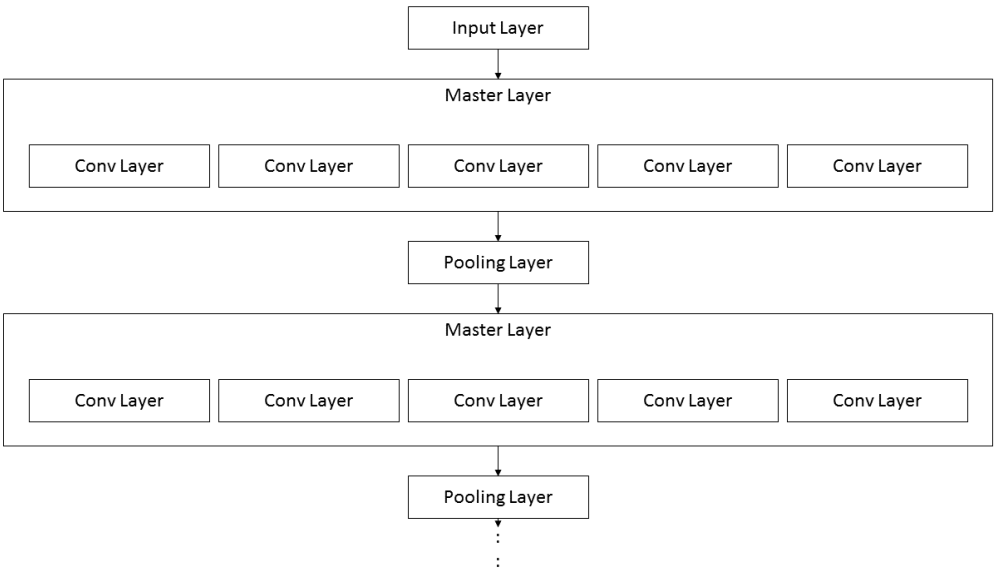
- Can Deep Convolutional networks be made rotation invariant
- What effect does the rotation interval have on result accuracy

5.3 Code Implementation

Part of this chapters contribution is the development of code for this method using the Caffe [65] machine learning framework. This subsection details the implementation of the method described in Section 5.1 and details use of the method within the Caffe framework. A copy of the code for this method can be found on GitHub at the following link https://github.com/ejbarrow/Caffe-Rotation_filters. The main files edited for this method are `"conv_layer"`, `"base_conv_layer"`, and `rot_max_layer` which can be found in the `"/src/caffe/layers"` and `"/include/caffe/layers"` folders.

Two implementation methods were tried before settling on an implementation. Both methods produce the same results but go about implementing this chapters method in slightly different ways. The first implementation tested followed an object orientation structure, editing the Convolutional Layer in Caffe to provide a Master layer which held a set of master weights and performed all merging and splitting of data matrices as well as weight updating. This master layer would then spawn convolutional layers inside itself and pass weights to each parallel layer contained and give each layer a rotation to use for its weights. The master layer could call forward and backwards propagation on each parallel layer and easily access the weights and deltas inside each layer. Figure 5.6 outlines how this would have looked when visualising the classes on the convolution section of a network.

Fig. 5.6 Object Orientated Implementation



This design was ultimately replaced with another implementation method which uses slightly more memory (or the same memory as before if memory is not released between layers in the previous method) as the previous method but yielded a speed increase of up to 10 times faster. This improved method uses just 1 convolutional layer to represent all the parallel layers in the parallel convolutional structure. Instead of duplicating filters across many layers which means each layer needs to run its own forward and backwards propagation as well as flattening of input data, the method duplicates the rotated filters inside its own weight structure by adding the duplicated rotated filters as extra filters in the network. On a 15 degree rotation network this meant that 24 versions of a filter were created, so the total number of filters in a layer would be equal to the number of rotations to be performed multiplied by the number of parallel layers which is 360 divided by the rotation interval.

Fig. 5.7 Filter Rotation Structure

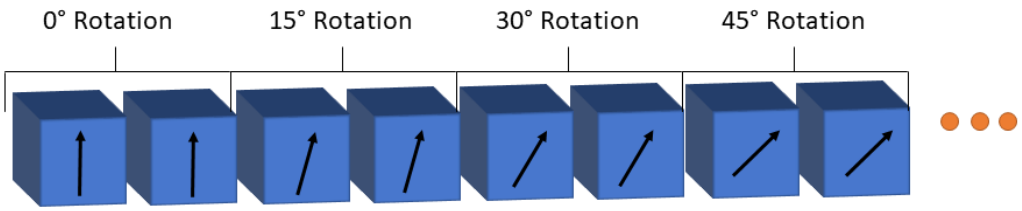


Figure 5.7 shows a visualisation of how filters are copied across the structure in groups and rotated, each group representing a rotation of the original group. A similar action is performed on the bias vector to duplicate the bias for each new filter that has been duplicated and rotated. This is also duplicated in groups so that, for example, the first bias in the vector is duplicated to match with the rotated versions for the filter it represents. As such the position of a copied bias in the vector (with an index starting at 0) is $originalindex + (numberof filters * copyID)$, where $copyID$ is the counter for the current copy (i.e. 1 to 23 in a 15 degree interval layer which has 24 versions of each filter).

Forward and backwards propagation can be run on the enlarged structure as if it was a singular layer. A pooling layer is used to manage the winner takes all pooling for the filters and to manage the splitting of deltas to be passed into the backwards propagation layer. In the network configuration the number of filters is defined for the convolutional layer in this method by multiplying the number of filters we want by the number of rotations.

5.4 Datasets

A number of different datasets were used for this chapters experiments to allow for observation between a more uniform dataset and some less uniform natural image datasets. The two datasets selected for this are the MNIST handwritten digits dataset (Section 4.1.2), and the CHARS74K natural images of characters and digits dataset (Section 3.1.2).

MNIST has no pre-processing performed on it, however CHARS74K will use the same dataset pre-processing as found in Section 3.1.2 to remove backgrounds and turn the image into a basic black and white structure. This will make classification easier and simpler than using colour images or even grey-scale images.

Each of these datasets will be transformed into 2 separate datasets. The first form of a dataset will be the dataset in its standard form, while the second form of the dataset will have the same training images, however the testing set will have random rotations throughout all the images. This is because of the need to train the network on a uniform or standard dataset but still be able to recognise rotations when not trained on a rotated set.

The datasets can be defined as follows with a dataset ID to be used in experiment tables:

- `MN_NR`: MNIST Dataset - Standard Training Set - Standard Test Set
- `MN_R`: MNIST Dataset - Standard Training Set - Rotated Test Set
- `CH_NR` : CHARS74K Dataset - Standard Training Set - Standard Test Set
- `CH_R` : CHARS74K Dataset - Standard Training Set - Rotated Test Set

The MNIST datasets are split into a Test set of 10,000 images and a Training set of 60,000 images. The CHARS74K dataset is split into a Test set of 1000 images and a Training set of 6,705 images. Figure 5.8 shows the distribution of random rotations across the test set of `MN_R` which is our MNIST dataset for testing on rotated images. Figure 5.9 shows the rotation distribution of random rotations on the test set for dataset `CH_R` made of CHARS74K images.

Fig. 5.8 `MN_R` Test Set Rotation Distribution

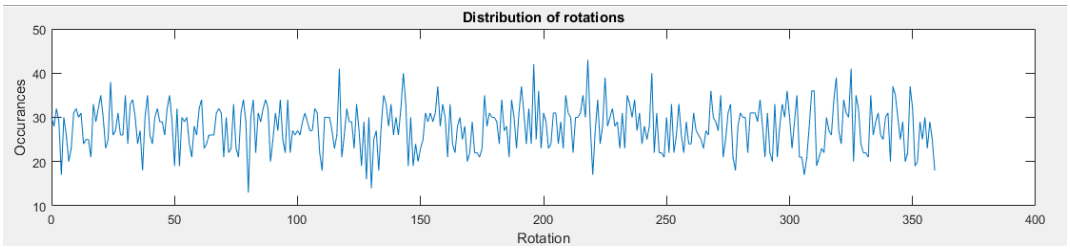
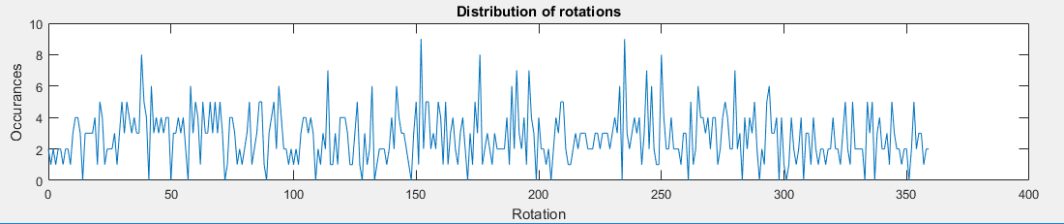


Fig. 5.9 CH_R Test Set Rotation Distribution



A standard neural network may be able to deal with a small amount of rotation within a couple of degrees of 0 due to some natural small rotations in the un-rotated datasets. After a certain point in the distribution of rotations it would become too difficult for a standard network to recognise the rotated images as it will not have been trained with them.

5.5 Experiments

5.5.1 Experiment Architectures

Each dataset pair has its own architecture which is unique to the dataset. The MNIST architecture is built off of the Caffe example for a convolutional neural network and altered to fit this method. This has been done as the MNIST example had already been fine-tuned for optimal results. The Chars74K architecture however is an altered version of the MNIST architecture, and has been enlarged to deal with larger images and more classes.

$$\alpha_t = \alpha_b * (1 + gamma * t)^{(-power)} \tag{5.7}$$

Architecture 1 (A_MNIST) is defined for MNIST as shown in figure 5.10. MNIST experiments use a base learning rate of 0.08 with a momentum of 0.9 and a weight decay of 0.0005. Batch normalisation allows for higher learning rates to be used so that a network may converge in fewer epochs. Learning rate decay is used with a gamma of 0.0001 and power of 0.75. Equation 5.7 shows how the learning rate is calculated for each iteration. In this equation t represents the iteration, α represents the learning rate, and α_b represents the base learning rate. Architecture 1 is based on a Caffe MNIST example network which has already been tuned to optimum values which will provide a great starting point for the network.

Architecture 2 (A_CHARS) is defined for CHARS74K shown in figure 5.11. CHARS74K experiments use a smaller base learning rate than MNIST of 0.01 due to the larger network size and input size. These experiments also use a momentum value of 0.9 and a weight decay of 0.0005. Learning rate decay is also used with a gamma of 0.0001 and power of 0.75.

The use of decay in these architectures help to reduce overfitting and getting stuck in local minima. The use of momentum also helps to prevent the networks getting stuck in local minima. Learning rate decay also allows for the network to make smaller steps towards the end of training to help it fine tune the minima that it has found without jumping around either side of the minima too much.

Fig. 5.10 `A_MNIST` Network Architecture

Input	Conv	BN	Pool	Conv	BN	Pool	FC (Conv acting as FC)	ReLU	BN	Output
28x28x1	Filters: 20 Filter Size: 7 Stride: 1		Filter Size: 2 Stride: 2	Filters: 50 Filter Size: 5 Stride: 1		Filter Size: 2 Stride: 1	(50*6*6)x250			250x10

Fig. 5.11 `A_CHARS` Network Architecture

Input	Conv	BN	Pool	Conv	BN	Pool	FC (Conv acting as FC)	ReLU	BN	FC	ReLU	BN	Output
50x50x1	Filters: 32 Filter Size: 3 Stride: 1		Filter Size: 2 Stride: 2	Filters: 32 Filter Size: 3 Stride: 1		Filter Size: 2 Stride: 2	(32*11*11)x256			256x1024			1024x62

5.5.2 Experiment list

Table 5.1 shows a list of experiments planned to test this sections method across several datasets and multiple rotation intervals.

Table 5.1 List of Experiments to be conducted

Experiment ID	Architecture ID	Dataset ID	Rotation Interval
M_NR_BENCH	A_MNIST	MN_NR	0
M_NR_15	A_MNIST	MN_NR	15
M_NR_30	A_MNIST	MN_NR	30
M_NR_60	A_MNIST	MN_NR	60
M_NR_90	A_MNIST	MN_NR	90
M_R_BENCH	A_MNIST	MN_R	0
M_R_15	A_MNIST	MN_R	15
M_R_30	A_MNIST	MN_R	30
M_R_60	A_MNIST	MN_R	60
M_R_90	A_MNIST	MN_R	90
C_NR_BENCH1	A_CHARS	CH_NR	0
C_NR_15	A_CHARS	CH_NR	15
C_NR_30	A_CHARS	CH_NR	30
C_NR_60	A_CHARS	CH_NR	60
C_NR_90	A_CHARS	CH_NR	90
C_R_BENCH	A_CHARS	CH_R	0
C_R_15	A_CHARS	CH_R	15
C_R_30	A_CHARS	CH_R	30
C_R_60	A_CHARS	CH_R	60
C_R_90	A_CHARS	CH_R	90

5.5.3 Experiment Resources

Due to the nature of the method, a large number of resources are required in comparison to a traditional convolutional layered network. There is an increase in memory used for experiments due to the duplication of filters, and this can be expected to be proportionate to the number of parallel layers/filters are running. There is then the increased CPU/GPU usage to manage duplicating filters, rotating filters, merging outputs, splitting deltas, merging weight updates, forward propagating a larger filter matrix, and backwards propagating the larger matrix. All of these factors lead to an increased run time depending on the rotation interval used.

5.6 Results

Table 5.2 shows the results of the conducted experiments for each version of the experiment as well as the average result for each experiment. Results are taken from the max test result achieved during the training process.

Table 5.2 Experiment Results

Experiment ID	V1	V2	V3	V4	V5	Average
M_NR_BENCH	0.9868	0.9906	0.9907	0.9902	0.9881	0.9893
M_NR_15	0.9855	0.9867	0.9852	0.9823	0.9838	0.9847
M_NR_30	0.9857	0.9862	0.9831	0.9859	0.9855	0.9853
M_NR_60	0.9851	0.9866	0.9863	0.9878	0.9862	0.9864
M_NR_90	0.9807	0.9831	0.9834	0.9812	0.9818	0.982
M_R_BENCH	0.4413	0.4416	0.4374	0.4367	0.4419	0.4398
M_R_15	0.961	0.9595	0.9678	0.9616	0.9649	0.963
M_R_30	0.9606	0.9497	0.9582	0.961	0.955	0.957
M_R_60	0.8545	0.8567	0.8385	0.8482	0.8783	0.8552
M_R_90	0.8889	0.8823	0.8934	0.8876	0.8824	0.8869
C_NR_BENCH	0.651	0.645	0.648	0.632	0.637	0.6426
C_NR_15	0.53	0.531	0.54	0.534	0.531	0.5332
C_NR_30	0.527	0.522	0.534	0.531	0.53	0.5288
C_NR_60	0.547	0.552	0.557	0.548	0.575	0.5558
C_NR_90	0.563	0.542	0.552	0.553	0.537	0.5494
C_R_BENCH	0.12	0.116	0.123	0.124	0.13	0.1226
C_R_15	0.486	0.49	0.498	0.48	0.481	0.487
C_R_30	0.41	0.427	0.429	0.448	0.434	0.4296
C_R_60	0.282	0.309	0.293	0.279	0.29	0.2906
C_R_90	0.229	0.234	0.246	0.245	0.238	0.2384

Table 5.3 shows the evaluation table which is a simplified and more condensed version of the results table. The results show the differences between the benchmark experiments and the rotation interval experiments. From the table it can be seen that for the non-rotated datasets the interval networks performed slightly worse (<1%) in the MNIST experiments, but much worse for the CHARS74K experiments. This could possibly be due to data loss or blurring during the rotation of the filters, but also due to numbers and characters in the dataset that look exactly the same in different orientations, such as a 9 and 6, or W and M, with this problem becoming More apparent with the CHARS74K experiments due to the increase in the number of classification categories. The benchmark network can deal with these numbers and letters better as it makes the assumption all inputs are upright, however

the rotation interval networks do not assume an orientation for the input, and as such with non-rotated data it performs worse.

However, with the two rotated test datasets for MNIST and CHARS74K it can be seen that the benchmark does spectacularly worse than the rotation interval network experiments. This is because a large number of input images are now rotated and the benchmark network has not learnt representations of the classifiers/filters in different orientations, whereas the rotation interval networks expect rotation and can apply its classifiers/filters in different orientations, and as such identify rotated input without the need to have trained on data with rotations.

Table 5.3 Experiment Evaluation Table

Dataset ID	Architecture ID	Benchmark	15°	30°	60°	90°
MN_NR	A_MNIST	0.9893	0.9847	0.9853	0.9864	0.982
MN_R	A_MNIST	0.4398	0.963	0.957	0.8552	0.8869
CH_NR	A_CHARS	0.6426	0.5332	0.5288	0.5558	0.5494
CH_R	A_CHARS	0.1226	0.487	0.4296	0.2906	0.2384

Rotation interval networks performed worse on the rotated dataset than they performed on the non-rotated dataset, giving a significant drop in accuracy results, with that accuracy dropping more for experiments with large rotation intervals. This is likely due to the network being trained on the non-rotated training set, which means the filter orientations directly apply to the upright orientations of the non-rotated test set. However with the rotated test set, a number of images may now sit in between rotation intervals, which is why we can see the experiments improve on the rotated test set when a smaller rotation interval is applied. This can be seen clearly in Figures 5.13 and 5.15 where results improve with smaller intervals.

In Table 5.3 the results show that for MNIST where the top rotation interval network (90°) had 0.007% worse accuracy on the non-rotated test set which could be seen as negligible, however with the rotated test set the best interval network (15°) achieved an improvement of 52.32% in accuracy accuracy compared to the benchmark network applied to the rotated test set. The CHARS74K results show that with the non-rotated dataset the top rotation interval network (60°) was 8.68% worse in test accuracy than the benchmark, however when the rotated test set comes into play the best interval network (15°) is 36.44% better at accurately predicting on a rotated test set than the benchmark.

Performing T-Tests on the 15°networks in comparison to their relative benchmarks shows the results are significant due to their small P-Values. Experiment M_NR_15 gives a P-Value of 0.006550982, M_R_15 gives a P-Value of 9.054E-10, C_NR_15 gives a P-Value of 4.64841E-06, and C_R_15 gives a P-Value of 8.881E-08.

Fig. 5.12 `MN_NR` - Average Test Accuracy for MNIST on a non-rotated test set

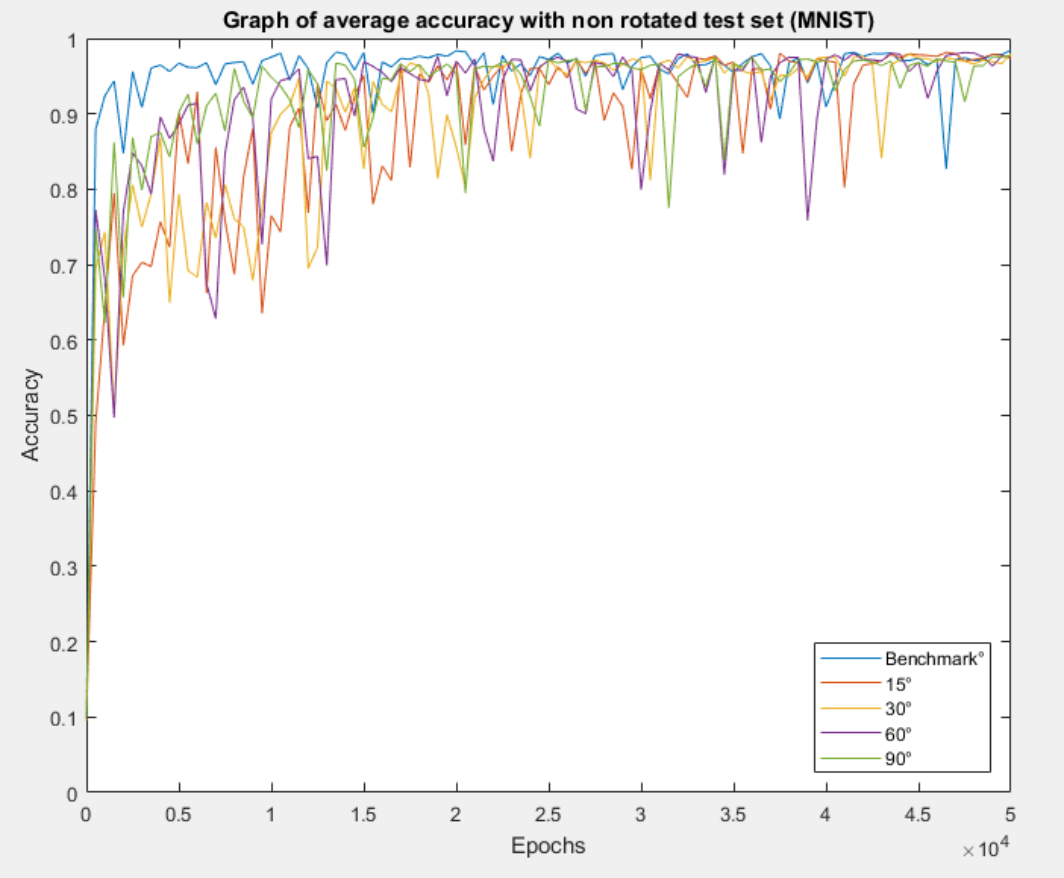


Figure 5.12 shows the test set accuracy during training for the non-rotated test set (`MN_NR`) for the benchmark and the rotation interval networks. From the figure it can be seen that the rotation interval networks take longer to converge than the benchmark network as well as typically performing worse than the benchmark network. Figure 5.13 shows the test set accuracy for the rotated test set (`MN_R`) where it can be seen that the rotation interval networks all out performed the benchmark network. The smaller the rotation interval that was set for an experiment was, it was found that the experiment accuracy increased. However for the 60 degree interval network it can be seen performing worse than the 90 degree network. All intervals smaller than 90 degrees that were tested needed to perform some form of interpolation, however for the 15 degree and 30 degree intervals the amount of rotations that needed interpolation was reduced as they contained rotations that fall on 90 degree rotations. The 60 degree interval has no rotations that sit on a 90 degree rotation, and as such all of the rotations (except 0°) for the 60 degree interval require interpolation. In this case it could be considered that the improvement from switching between 90 and 60 degree intervals was

smaller than the the loss of accuracy from having to do extra interpolation on the 60 degree interval network.

Fig. 5.13 `MN_R` - Average Test Accuracy for MNIST on a rotated test set

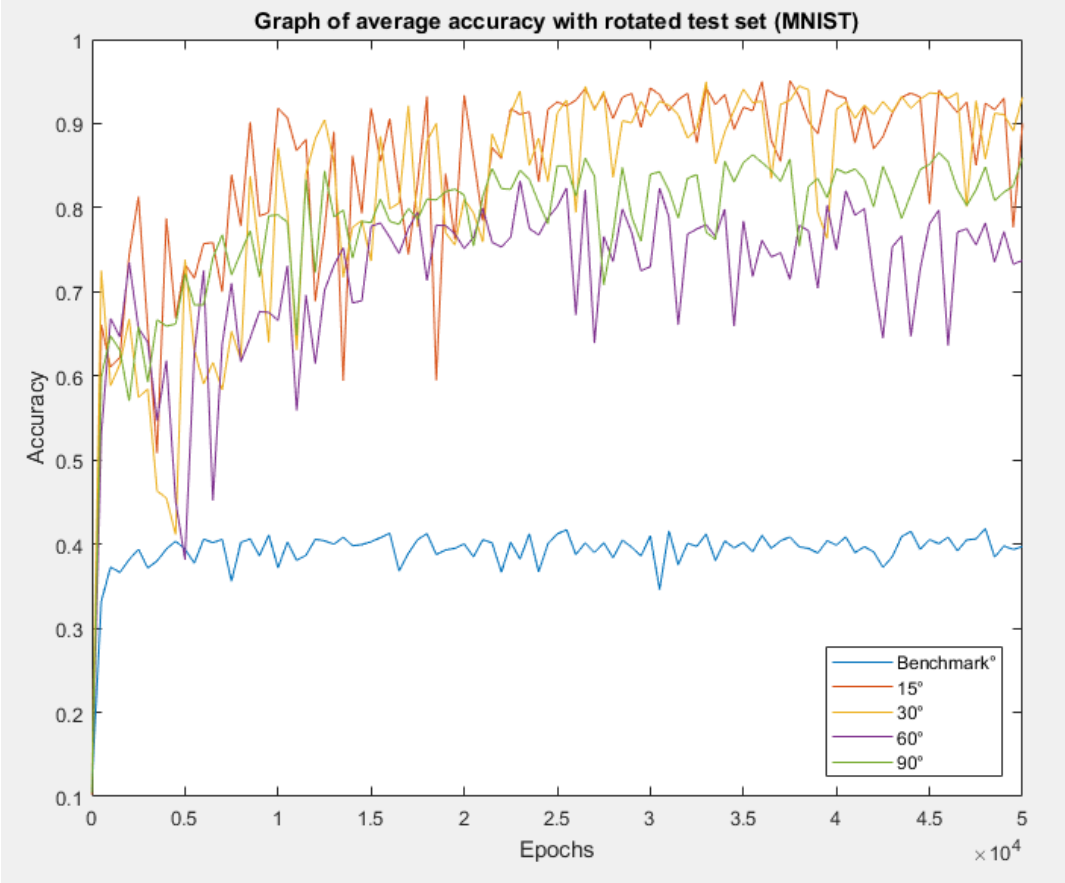


Fig. 5.14 CH_NR - Average Test Accuracy for CHARS74K on a non-rotated test set

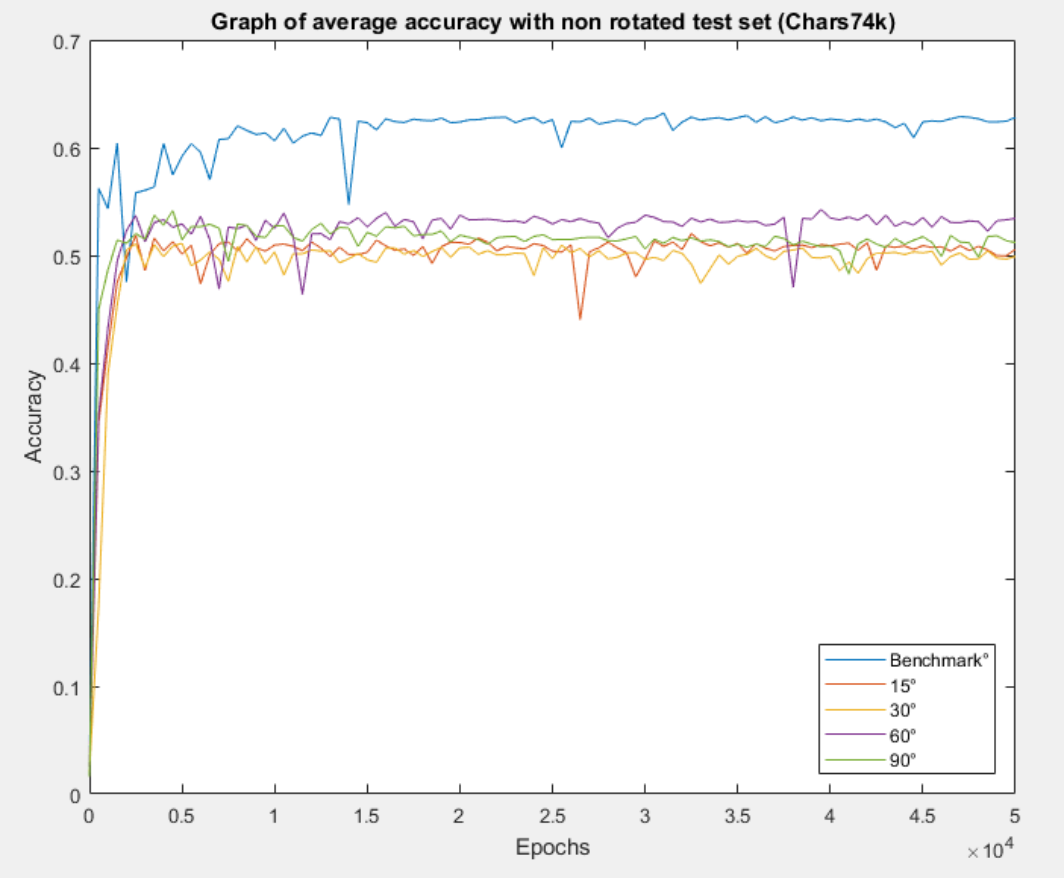


Figure 5.14 shows a graph of test set accuracy during training for the non-rotated test set. The graph shows that the benchmark network performed much better for the CHARS74K dataset than the rotation interval networks. The rotation interval networks are very close to each other and could be considered almost the same degree of accuracy within the realm of random initialisation. Figure 5.15 shows much better results for the rotation interval networks where the benchmark has performed badly in comparison. It can clearly be seen in the figure that the smaller the rotation interval used the better the networks have performed on the CHARS74K dataset, with a 15 degree interval topping the graph. Compared to the MNIST experiments the 60 degree network beat the 90 degree network in the CHARS74K experiments, which is the opposite case to what happened with the MNIST results. This could be explained by the larger input images, where smaller incremental values become more important as an image is enlarged.

The graphs of results presented show the rotation interval networks jump around a lot in comparison to the benchmark network even though they use the same learning rates. This could be due to the weight updates in a winner takes all method causing larger updates than

would usually be seen in a normal convolutional network, and as such may suggest that the method could benefit from a reduced learning rate compared to the normal convolutional benchmark counterpart.

Fig. 5.15 CH_R - Average Test Accuracy for CHARS74K on a rotated test set

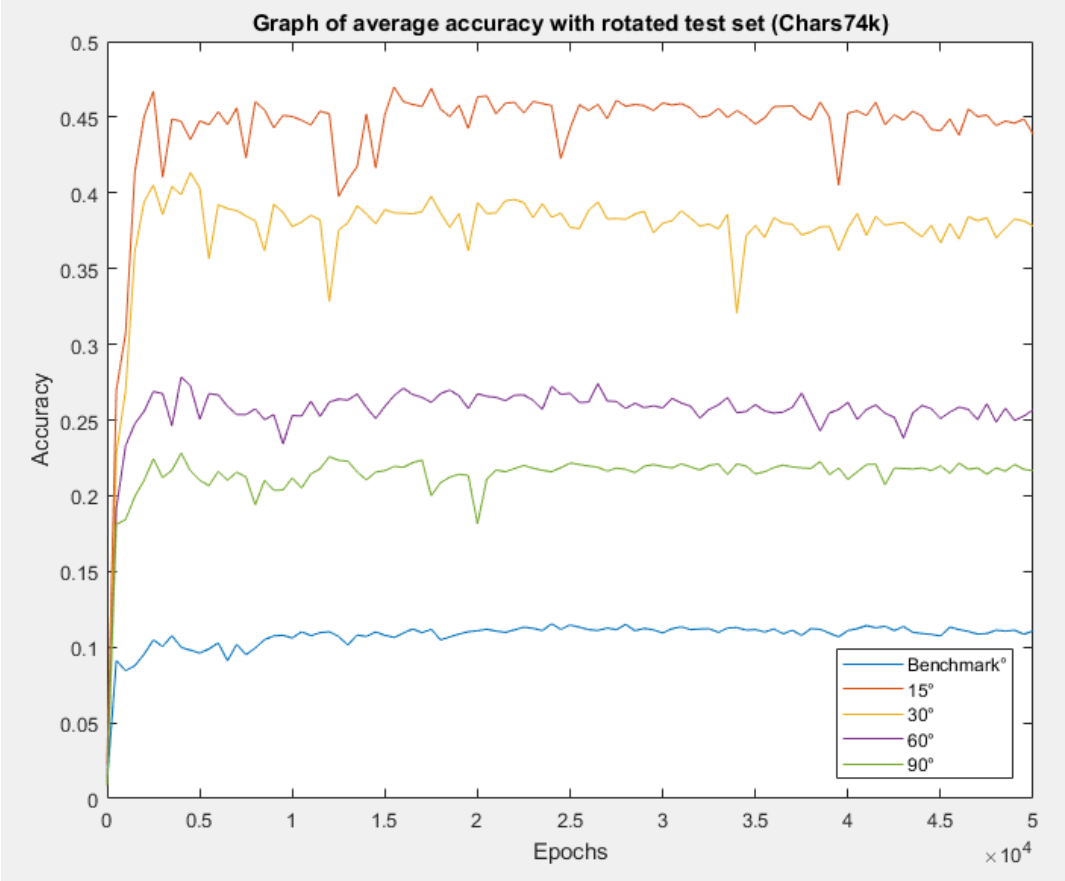


Figure 5.16 shows a representation of filters from the first convolutional layer in both the benchmark network and a 15 degree interval network for MNIST. The benchmark filters can be seen as quite complex compared to the filters of CHARS74K (Figure 5.17) due to the larger size of filter. The filter representations for the 15 degree interval network are interesting in comparison to the benchmark network filters, as they seem to use less data to represent features and patterns as well as seeming more simplistic.

Fig. 5.16 A representation of the filters in Convolutional Layer 1 (MNIST)

Figure 5.17 shows a representation of filters from the first convolutional layer in both the benchmark network and a 15 degree interval network for the CHARS74K experiments. There are some minor differences between the two sets of filters. In the benchmark filters it can be seen that almost all filters are different, however the 15 degree interval network has repeated a number of filters, indicating that perhaps a rotation invariant network does not need as many filters as filter representations that are similar do not need to be learnt if they are the same in different orientations. Another possibility for the repeated filters could be because in a 3 by 3 circular filter there are only 5 working pixels, giving a very finite number of possible filters compared to the benchmark.

Fig. 5.17 A representation of the filters in Convolutional Layer 1 (CHARS74K)

5.7 Conclusion

It can be concluded from the results that this method works at producing rotation invariant convolutional neural networks. Results showed a clear improvement over the benchmark convolutional network when a rotated test set is used, with an improvement of 52.32% in accuracy for the MNIST dataset and 36.44% accuracy on the CHARS74K dataset when comparing the 15 degree network and the benchmark network. On a non-rotated test set the results came out slightly worse than the benchmark result, but with more refined networks and datasets (such as MNIST) the loss of accuracy was negligible. This could be due to a number of factors, from classes that look similar when rotated, to data loss and blurring during the rotation process. With datasets that were more complex with less refined networks the advantage of applying the rotation interval networks to rotated data was more apparent.

It was also shown that the accuracy of a network used with a rotated test set increased as the rotation interval of that network was made smaller, with smaller intervals generally showing better results than their larger interval counterparts. Further work on the method could look into using smaller rotation intervals as well as routing networks.

5.8 Pre-Rotation Filter Scaling

This section builds on the work from the previous section by exploring the effect of scaling filters before and after rotation on network accuracy. The motivation for this is that filters in some networks can be particularly small which can make them hard to rotate without a loss of data, so exploring whether scaling a filter up in size before being rotated and then scaling it back down may be a way to combat any loss or blurring of data during rotation. It would also be possible to use more complex interpolation methods in an attempt to reduce data loss from rotation of the filters.

5.8.1 Methodology

To test the method in this section a number of experiments will be conducted using the scaling method. These experiments will mirror experiments conducted in the begginng of Chapter 5, as to allow the previous experiments to act as benchmark results for comparison. A reduced amount of experiments will be run as we have already found that 15 and 30 degrees give the best results for our networks.

5.8.1.1 Research Questions

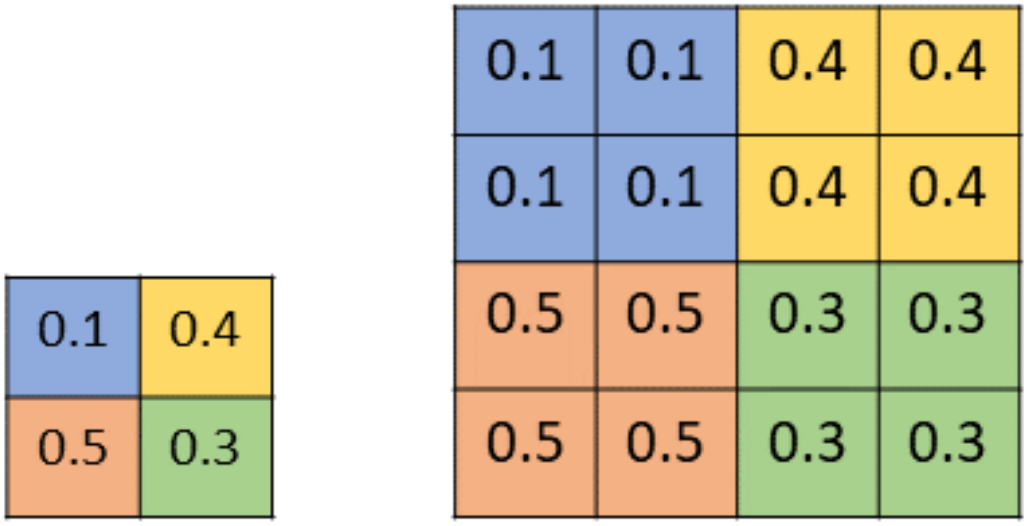
The research question for this section is whether a rotation invariant network can be improved by scaling filters before and after rotation in an attempt to reduce data loss and or blurring.

5.8.2 Methods

The base methods of this alteration are given in Section 5.1. To build on the base method the rotation method is altered to allow for scaling before and after rotation. For this thesis a scaling percentage of double the original size will be used to scale filters for rotation in an attempt to minimise data loss or blurring. For example a 3 by 3 filter will be scaled to 6 by 6 before rotation, and then scaled back to 3 by 3 after rotation.

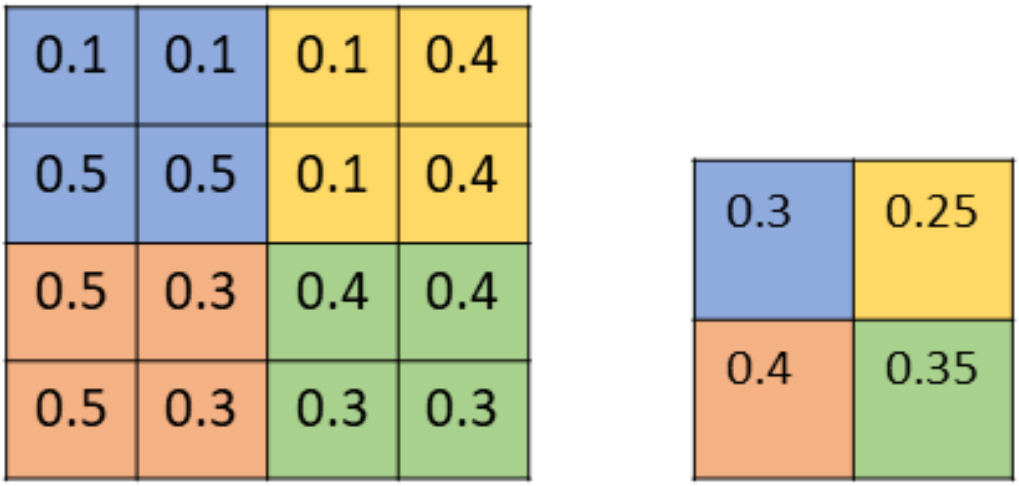
To upscale a filter a duplication approach is taken. Each weight is duplicated into a block of four weights and placed in a new position in the enlarged weight matrix. Figure 5.18 shows an example of this up-scaling process with a 2 by 2 grid, with each colour showing the duplication and new locations of each weight.

Fig. 5.18 Example Scaling of a 2 by 2 filter to 4 by 4



To downscale a filter after rotation an average pooling approach is taken, with groups of 4 weights being combined into a singular weight in the smaller weight matrix using an average of the 4 weights. Figure 5.19 shows an example of this down-scaling process with coloured cells showing the 4 weights being combined and their result in the smaller matrix.

Fig. 5.19 Example Down-scaling of a 4 by 4 filter to 2 by 2



5.8.3 Code Implementation

This section builds on the code of the previous sections where the code can be found at the same GitHub link as detailed in Section 5.3. A flag was added to the convolutional layer parameter set that allows for scaling of the filters to be switched on and off easily. If switched on the layer will scale a filter up by 100% before rotating the filters and then downscale back to the original size.

5.8.4 Experiments

The datasets used for this series of experiments and how they are processed are detailed in the dataset section of the previous methods section (Section 5.4). With the results of the previous chapter it is shown that 15 degree and 30 degree networks give the best accuracy results for the two datasets tested. As these are the best networks this method extension will run a reduced amount of experiments from the original method for just these two rotations. This not only helps reduce the amount of experiments that need to be run, but also the amount of time and resources it takes to run those experiments as discussed in Section 5.5.3. The experiments can also be run for just the rotated test set as this will likely show more clearly any improvements if any are to be found.

The same architectures defined in Section 5.5.1 will be used for these experiments with the only difference being that the filter scaling parameter is switched on for these experiments.

Table 5.4 List of Experiments to be conducted with Scaled Filters

Experiment ID	Architecture ID	Dataset ID	Rotation Interval
M_R_15_Scale	A_MNIST	MN_R	15
M_R_30_Scale	A_MNIST	MN_R	30
C_R_15_Scale	A_CHARS	CH_R	15
C_R_30_Scale	A_CHARS	CH_R	30

5.8.5 Results

Table 5.5 shows the results of the conducted experiments for each version of the experiment as well as the average result for each experiment.

Table 5.5 Experiment Results

Experiment ID	V1	V2	V3	V4	V5	Average
M_R_15_Scale	0.9652	0.9601	0.9694	0.9613	0.9547	0.9621
M_R_30_Scale	0.954	0.9583	0.9515	0.9541	0.9548	0.9545
C_R_15_Scale	0.443	0.469	0.427	0.447	0.458	0.4488
C_R_30_Scale	0.394	0.382	0.411	0.395	0.389	0.3942

Table 5.6 shows the evaluation table which is a simplified and more condensed version of the results table. In the table we can see results for the different datasets against the different rotation intervals and benchmark. The benchmarks are copied from the related experiments in Section 5.6 for easier side by side comparison.

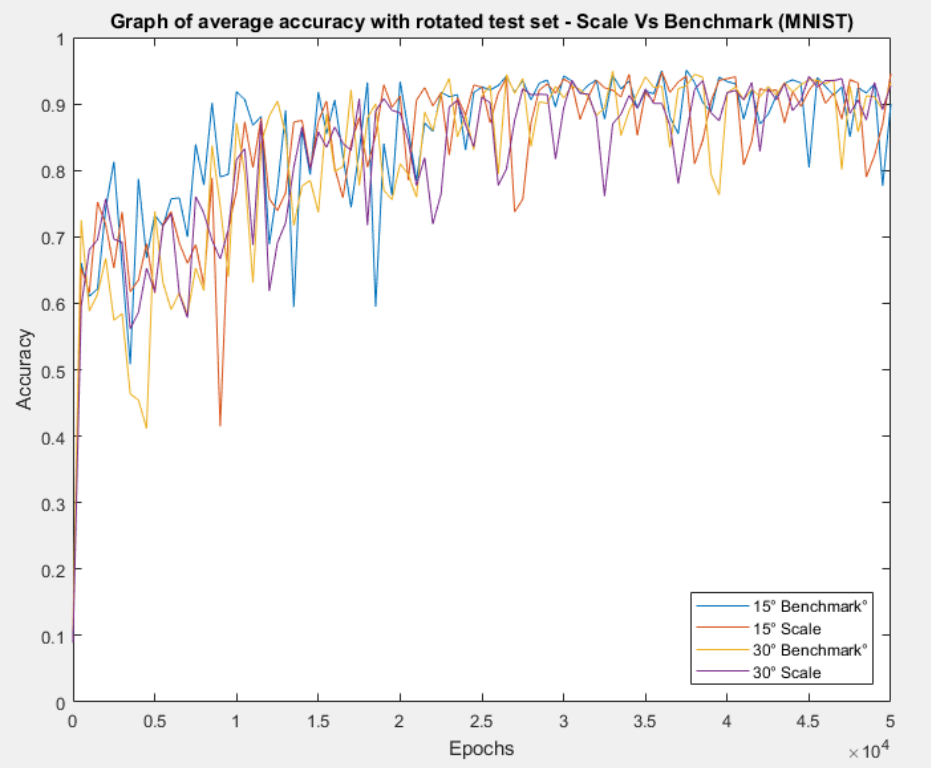
Table 5.6 Experiment Evaluation Table

Dataset ID	Architecture ID	15°Benchmark	15°	30°Benchmark	30°
MN_R	A_MNIST	0.963	0.9621	0.957	0.9545
CH_R	A_CHARS	0.487	0.4488	0.4296	0.3942

Examining the results of the MNIST experiments it can be seen that the 15 and 30 degree networks using the scaling method had a very similar but slightly worse average accuracy than the benchmark network. Figure 5.20 shows a graph of test set result accuracy during training, where it can be seen that results between the benchmarks and their relative scaling method networks were very close for the whole training period.

Examining results of the CHARS74K experiments show that for a 15 degree interval the scaling method produced an average result with 03.82% worse accuracy than the benchmark 15 degree network not using the method. The results also show an average drop of 03.54%

Fig. 5.20 MNIST Results Graph - Scale experiments



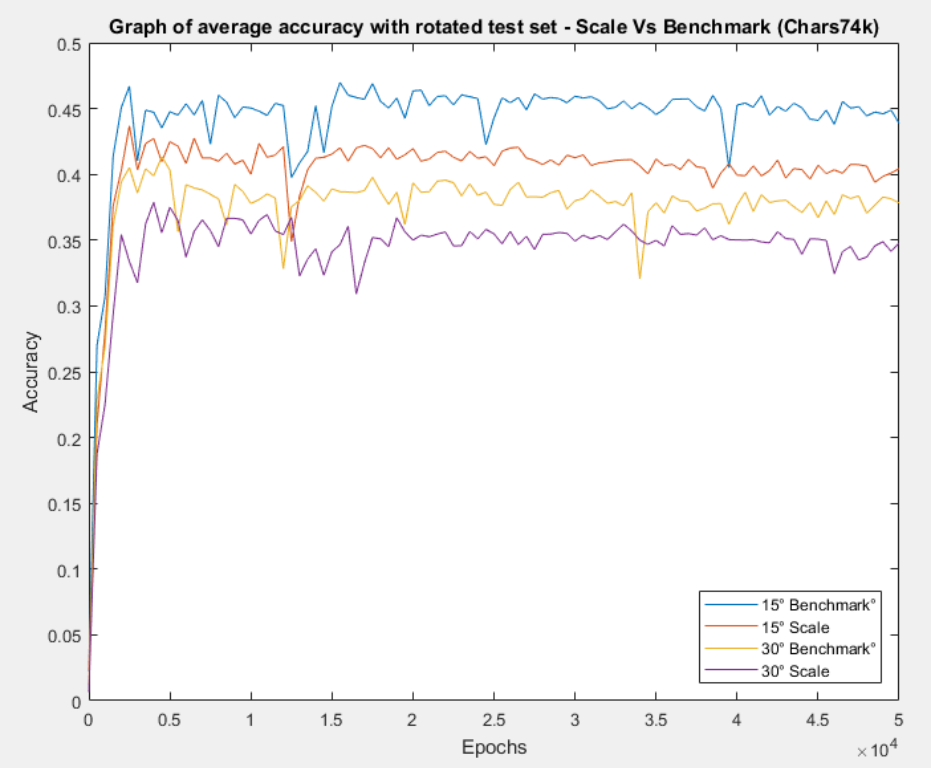
accuracy on the 30 degree interval networks. Figure 5.21 shows the test set accuracy during training. These results show a clearer distinction between the benchmark and the scale method networks compared to the MNIST experiments, possibly due to the extra complexity of the network and problem compared to the uniform and refined MNIST experiments.

Considering the results of both the MNIST and CHARS74K experiments it can be seen that the scaling method presented in this chapter is not useful in attempts to improve accuracy and reduce data loss/blurring during the rotation stage of the rotation invariant convolutional network. A possible cause for the results performing slightly worse than the benchmarks could show that the method is causing loss or blurring of more data than the rotation method created standalone. The extra data blurring and loss would likely be caused by the interpolation method (average pooling) used to scale down a filter.

5.8.6 Conclusion

To conclude, it has been found that the scaling method presented in this chapter produced worse results than benchmark methods. The method attempted to reduce data loss and blurring during the interpolation of filters during rotation, however the method produced

Fig. 5.21 CHARS74K Results Graph - Scale experiments



more data loss and blurring possibly due to the use of average pooling during filter down-scaling. On average it was found that results had a 0.17% less accuracy when applied to the MNIST dataset and a 3.68% worse accuracy applied to the CHARS74K dataset. Further work could be performed to attempt different and more accurate interpolation methods for this scaling method, however it could also be found that improving the original rotation method may yield better results than attempting to improve on the scaling interpolation method.

Chapter 6

Conclusions

6.1 Discussion

This thesis has discovered and evaluated a number of methods for deep learning which either improve the performance of deep learning itself or add rotation invariance to a network. In Chapter 3 dropout was discussed as a way to improve deep network performance and was further improved through the use of selective dropout as discussed in Chapter 4. Chapter 5 looked at creating a rotation invariant convolutional neural network which Chapter 5.8 attempted to build upon to further increase that performance.

The objectives discussed in Section 1.1.3 have been met by this thesis. This thesis has looked at how rotation effects the results of deep learning methods and has created a method for solving rotation invariance by improving upon the convolutional neural network method. This thesis also improved on the Dropout method with the use of selective dropout which could be used for improving results of deep neural networks.

Chapter 3 looked at the use of dropout with deep neural networks. Three research questions were asked in this chapter, all of which were answered though the chapters experiments. It was shown that dropout increased results for natural character recognition against a benchmark, that the dropout method can improve results even more with an extended training time, and that the the network architecture size did not improve or decrease the effectiveness of the dropout method.

In Chapter 4 It was found that the dropout methodology used to reduce overfitting in deep neural networks could be improved by selectively choosing neurons to be switched off during training rather than randomly dropping neurons as is typical with traditional uniform dropout. It was also found that providing a bad method for selecting neurons for dropping can dramatically reduce the accuracy of a neural network. The best of the methods tested for selective dropout was to give high drop probability to neurons with a small output variance

giving an average improvement of 1.17%. This answered the sections research questions of whether dropout could be improved by selectively choosing drop neurons, as well as choosing which method would be best to choose those neurons.

For introducing rotation invariance into a convolutional network it was found in Chapter 5 that rotation interval convolutional layers could be used to allow filters in a convolutional layer to become co-variant to rotation and the network as a whole to become approximately invariant when these rotation interval convolutional layers were paired with a rotation interval fully connected layer. The method dramatically improved results on rotated data, however saw slightly worse results when purely upright data was presented in comparison to a benchmark network trained to only recognise upright images. A number of reasons for this reduction in accuracy could be put down to data loss or blurring in the rotation algorithm, or to classifiers that are the same when orientations are removed (such as the letters W and M). These results answered the research question of whether a deep convolutional network could be made rotation invariant.

The last research question of Chapter 5 looked at whether the rotation interval had an effect on the accuracy of the network results. It was found that the level of invariance obtained by a network was related to the rotation interval selected for a network, with a network having a higher level of invariance, and an higher accuracy on rotated data, when a smaller rotation interval is set, improving result accuracy with a smaller interval. An attempt to improve on these results was made using a scaling method before and after filter rotation to reduce data loss and blurring, however it was found that this caused more data blurring or loss and led to worse results than not using the scaling method.

The results of this thesis are most useful for not only improving existing models using the traditional dropout method, but for helping models identifying objects in images that are often rotated or have no rotation orientation in the image, such as top down photography or space imagery. The methods presented would reduce the need to train neural network models on large datasets with artificial rotations permutated into them.

6.2 Future Work

6.2.1 Scale Invariant Convolutional Neural Networks

Using the ideas from Rotational Filters in Convolutional Neural Networks (Chapter 5) it would theoretically be possible to create a scale invariant network in a similar manner by scaling copies of filters to allow it to detect features at different sizes. The filters outputs can then be scaled back and pooled to create a single filter output.

While an interesting idea it would be slightly more difficult to implement than rotation due to the varying sizes in filters, and it would be an unwise idea currently to combine this method with rotation invariance at the same time, as for every filter scale performed there would need to be a set of rotations. Meaning for N filters we would need NxRxS filters in the whole layer, where R represents the number of rotations tested, and S representing the number of scales to be tested. This while not impossible, would take a lot of processing power and extend the time it takes to train as well as use the network, though as computers become more powerful could be a possible way of combining invariant networks.

6.2.2 Selective Dropout for Weights

The dropout methodology has been applied to weights for neurons in a layer, whereby a percentage of weights are dropped from each neuron instead of switching of an entire neuron [135]. The authors of this method call it Drop Connect. It would theoretically be possible to implement a form of selective dropout (Chapter 4) with weights.

Not all selective methods applied in this thesis would be suitable for ranking individual weights, such as output variance as this is a product found from all weights attached to a node. It should be possible to implement the weight change method with individual weights as the method for whole neurons uses an average of the statistic for all its connected weights. It could also be possible to implement the average weight method by giving a higher drop percentage to weights with a small or large value.

6.2.3 Rotation Interpolation

Results in Chapter 5 showed great improvements in accuracy for rotated data in comparison to accuracy of networks not using the rotation interval method. However in cases of upright data the rotation interval networks were out performed by their traditional convolutional network counterparts. A possible improvement to this method could be to examine different methods of rotation interpolation beyond the scaling method attempted in Chapter 5.8.

6.2.4 Routing Networks

Results in Chapter 5 showed that on non-rotated data the best network to use was a benchmark network set up not to recognise rotations in images. However this is not useful for datasets that contain images in a variety of orientations. A possible research avenue to further this work could be to look into the use of a router network that identifies if an image is rotated and pass it to the applicable network based on the orientation found. This could however

be quite difficult and would require training for each dataset used, as well as being counter productive if the network can identify a rotation degree as it would be simpler to then correct the image rotation before passing it to a network.

6.2.5 Galaxy Images

One of the benefits discussed in Chapter 5 was the ability to work with images that contain no default orientation. As such, an application of the method to galaxy images (such as the Galaxy zoo dataset [87]) would be a interesting extension of current work due to the nature of orientation in space.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Abdel-Hakim, A. E. and Farag, A. A. (2006). Csift: A sift descriptor with color invariant characteristics. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1978–1983. IEEE.

[3] Abdi, H. and Williams, L. J. (2010). Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459.

[4] Amazon (2017). Amazon Go. https://www.amazon.com/b?node=16008589011.

[5] American Honda Motor Co (2017). ASIMO by Honda. http://asimo.honda.com/.

[6] Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(1):54–65.

[7] Ba, J. and Frey, B. (2013). Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092.

[8] Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. (2013). Collaborative hyperparameter tuning. In *International Conference on Machine Learning*, pages 199–207.

[9] Barrow, E., Eastwood, M., and Jayne, C. (2016). Selective dropout for deep neural networks. In *Neural Information Processing*, Lecture Notes in Computer Science. Springer International Publishing.

[10] Barrow, E., Jayne, C., and Eastwood, M. (2015). Deep dropout artificial neural networks for recognising digits and characters in natural images. In Arik, S., Huang, T., Lai, W. K., and Liu, Q., editors, *Neural Information Processing*, volume 9492 of *Lecture Notes in Computer Science*, pages 29–37. Springer International Publishing.

[11] Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. (2016). Deepmind lab. *arXiv preprint arXiv:1612.03801*.

[12] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127.

[13] Blair, A. (2015). Variations on Backpropagation. http://www.cse.unsw.edu.au/~cs3411/15s1/lect/1page/13a_Backprop.pdf.

[14] Blumenstein, M., Liu, X. Y., and Verma, B. (2004). A modified direction feature for cursive character recognition. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 4, pages 2983–2987. IEEE.

[15] Booz Allen Hamilton, I. (2014). National data science bowl.

[16] Boston Dynamics Inc (1992). Boston dynamics. https://www.bostondynamics.com/.

[17] Brunelli, R. and Poggio, T. (1993). Face recognition: Features versus templates. *IEEE transactions on pattern analysis and machine intelligence*, 15(10):1042–1052.

[18] Buckley, J. J. and Hayashi, Y. (1994). Fuzzy neural networks: A survey. *Fuzzy sets and systems*, 66(1):1–13.

[19] Buscema, M. (1998). Back propagation neural networks. *Substance use & misuse*, 33(2):233–270.

[20] Cellan-Jones, R. (2014). Stephen Hawking warns artificial intelligence could end mankind.

[21] Chen, F.-C. (1990). Back-propagation neural networks for nonlinear self-tuning adaptive control. *Control Systems Magazine, IEEE*, 10(3):44–48.

[22] Chen, H. and Murray, A. F. (2003). Continuous restricted Boltzmann machine with an implementable training algorithm. In *Vision, Image and Signal Processing, IEE Proceedings-*, volume 150, pages 153–158. IET.

[23] Chen, Y.-C. and Teng, C.-C. (1995). A model reference control structure using a fuzzy neural network. *Fuzzy Sets and Systems*, 73(3):291–312.

[24] Cireşan, D., Meier, U., Masci, J., and Schmidhuber, J. (2012). Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338.

[25] Cireşan, D. and Schmidhuber, J. (2013). Multi-column deep neural networks for offline handwritten chinese character classification. *arXiv preprint arXiv:1309.0261*.

[26] Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2011). Convolutional neural network committees for handwritten character classification. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1135–1139. IEEE.

[27] Ciresan, D. C., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745.

[28] Clarifai (n.d.). Clarifai / technology. https://www.clarifai.com/technology.

[29] Cortes, C. and Vapnik, V. (1995). Support vector machine. *Machine learning*, 20(3):273–297.

[30] Dahl, G., Mohamed, A.-r., Hinton, G. E., et al. (2010). Phone recognition with the mean-covariance restricted Boltzmann machine. In *Advances in neural information processing systems*, pages 469–477.

[31] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE.

[32] Dallas, G. (2013). Principal component analysis 4 dummies: Eigenvectors, eigenvalues and dimension reduction.

[33] Danker, A. J. and Rosenfeld, A. (1981). Blob detection by relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(1):79–92.

[34] de Campos, T. E., Babu, B. R., and Varma, M. (2009). Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*.

[35] DeepLearning.net (2008). Restricted Boltzmann machines (RBM).

[36] Déniz, O., Bueno, G., Salido, J., and De la Torre, F. (2011). Face recognition using histograms of oriented gradients. *Pattern Recognition Letters*, 32(12):1598–1603.

[37] DiCarlo, J. J. and Cox, D. D. (2007). Untangling invariant object recognition. *Trends in Cognitive Sciences*, 11(8):333 – 341.

[38] Dieleman, S. (2015). Classifying plankton with deep neural networks.

[39] Dieleman, S., Willett, K. W., and Dambre, J. (2015). Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450(2):1441–1459.

[40] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2013). Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*.

[41] Duyck, J., Lee, M. H., and Lei, E. (2014). Modified dropout for training neural network. Paper Published online.

[42] Edelman, S. (1999). *Representation and recognition in vision*. MIT press.

[43] Epitropakis, M., Plagianakos, V., and Vrahatis, M. (2006). Higher-order neural networks training using differential evolution. In *International Conference of Numerical Analysis and Applied Mathematics, Wiley-VCH, Crete, Greece*, pages 376–379.

[44] Facebook Inc (2004). Facebook. https://www.facebook.com/.

[45] Farah, M. J. and Hammond, K. M. (1988). Mental rotation and orientation-invariant object recognition: Dissociable processes. *Cognition*, 29(1):29 – 46.

[46] Felleman, D. J. and Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1(1):1–47.

[47] Fergus, R., Leeand, H., Ranzato, M., Salakhutdinov, R., Taylor, G., and Yu, K. (2012). Deep learning methods for vision.

[48] Ferrari, S. (2012). Image segmentation. http://cvg.ethz.ch/teaching/2010fall/compvis/lecture/segmentation.pdf.

[49] Franzius, M., Wilbert, N., and Wiskott, L. (2008). Invariant object recognition with slow feature analysis. In *Artificial Neural Networks-ICANN 2008*, pages 961–970. Springer.

[50] Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

[51] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

[52] Gibiansky, A. (2014). Convolutional neural networks. http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/.

[53] Giles, C. L. and Maxwell, T. (1987). Learning, invariance, and generalization in high-order neural networks. *Applied optics*, 26(23):4972–4978.

[54] Goh, A. (1995). Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 9(3):143–151.

[55] Gonzalez, D. M., Volpi, M., and Tuia, D. (2016). Learning rotation invariant convolutional filters for texture classification. *CoRR*, abs/1604.06720.

[56] Google Inc (2013). Google Glass. https://www.x.company/glass/.

[57] Google Inc (2014). Google Goggles. https://play.google.com/store/apps/details?id=com.google.android.apps.unveil&hl=en_GB.

[58] Grauman, K. and Leibe, B. (2011). Visual object recognition. *Synthesis lectures on artificial intelligence and machine learning*, 5(2):1–181.

[59] Guo, G., Li, S. Z., and Chan, K. (2000). Face recognition by support vector machines. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 196–201. IEEE.

[60] Guo, Y., Bennamoun, M., Sohel, F., Lu, M., and Wan, J. (2014). 3d object recognition in cluttered scenes with local surface features: A survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(11):2270–2287.

[61] Hinton, G., Deng, L., Yu, D., Dahl, G., rahman Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*.

[62] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554.

[63] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[64] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[65] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.

[66] Jolliffe, I. (2002). *Principal component analysis*. Wiley Online Library.

[67] Jung, S. and Kim, S. S. (2007). Hardware implementation of a real-time neural network controller with a dsp and an fpga for nonlinear systems. *Industrial Electronics, IEEE Transactions on*, 54(1):265–271.

[68] Karic, M. and Martinovic, G. (2013). Improving offline handwritten digit recognition using concavity-based features. *International Journal of Computers Communications & Control*, 8(2):220–234.

[69] Karnin, E. D. (1990). A simple procedure for pruning back-propagation trained neural networks. *Neural Networks, IEEE Transactions on*, 1(2):239–242.

[70] Karpathy, A. (2016). Cs231n: Convolutional neural networks for visual recognition. *Neural networks*, 1.

[71] Ke, Q. and Li, Y. (2014). Is rotation a nuisance in shape recognition? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4146–4153.

[72] Kheradpisheh, S. R., Ganjtabesh, M., and Masquelier, T. (2015). Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition. *CoRR*, abs/1504.03871.

[73] Kotsiantis, S. B., Zaharakis, I., and Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24.

[74] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012a). Imagenet classification with deep convolutional neural networks. In Bartlett, P., Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 25*, pages 1106–1114. Curran Associates.

[75] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012b). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[76] Ku, C.-C. and Lee, K. Y. (1995). Diagonal recurrent neural networks for dynamic systems control. *Neural Networks, IEEE Transactions on*, 6(1):144–156.

[77] Kuo, R. J., Chen, C., and Hwang, Y. (2001). An intelligent stock trading decision support system through integration of genetic algorithm based fuzzy neural network and artificial neural network. *Fuzzy sets and systems*, 118(1):21–45.

[78] Lawrence, S., Giles, C. L., Tsoi, A. C., and Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113.

[79] Le Callet, P., Viard-Gaudin, C., and Barba, D. (2006). A convolutional neural network approach for objective video quality assessment. *Neural Networks, IEEE Transactions on*, 17(5):1316–1327.

[80] Le Cun, B. B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer.

[81] Le Roux, N. and Bengio, Y. (2008). Representational power of restricted Boltzmann machines and deep belief networks. *Neural Comput.*, 20(6):1631–1649.

[82] LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361:310.

[83] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[84] Leng, G., McGinnity, T. M., and Prasad, G. (2005). An approach for on-line extraction of fuzzy rules using a self-organising fuzzy neural network. *Fuzzy sets and systems*, 150(2):211–243.

[85] Lindeberg, T. (2012). Scale invariant feature transform. *Scholarpedia*, 7(5):10491.

[86] LinkKratzert, F. (2016). Understanding the backward pass through batch normalization layer. https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html.

[87] Lintott, C. J., Schawinski, K., Slosar, A., Land, K., Bamford, S., Thomas, D., Raddick, M. J., Nichol, R. C., Szalay, A., Andreescu, D., Murray, P., and Vandenberg, J. (2008). Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey. *Monthly Notices of the Royal Astronomical Society*, 389:1179–1189.

[88] Liu, B., Wang, M., Foroosh, H., Tappen, M., and Pensky, M. (2015). Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814.

[89] Liu, H., Guo, Q., Xu, M., and Shen, I.-F. (2008). Fast image segmentation using region merging with a k-nearest neighbor graph. In *2008 IEEE Conference on Cybernetics and Intelligent Systems*, pages 179–184.

[90] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee.

[91] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

[92] Luo, S., Mouy, W., Li, M., Althoefer, K., and Liu, H. (2014). Rotation and translation invariant object recognition with a tactile sensor. In *SENSORS, 2014 IEEE*, pages 1030–1033. IEEE.

[93] Maguire, J. (2014). Robot security guard.

[94] Maguire, L. P., Roche, B., McGinnity, T. M., and McDaid, L. (1998). Predicting a chaotic time series using a fuzzy neural network. *Information Sciences*, 112(1):125–136.

[95] Marr, D. and Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217.

[96] Mehrotra, R., Nichani, S., and Ranganathan, N. (1990). Corner detection. *Pattern Recognition*, 23(11):1223–1233.

[97] Mika, S., Rätsch, G., Weston, J., Schölkopf, B., Smola, A. J., and Müller, K.-R. (2000). Invariant feature extraction and classification in kernel spaces. In *Advances in neural information processing systems*, pages 526–532.

[98] Miller III, W. T. (1994). Real-time neural network control of a biped walking robot. *Control Systems, IEEE*, 14(1):41–48.

[99] Nielsen, M. A. (2015). Neural networks and deep learning.

[100] ONLINE, F. T. (2015). Papo piebald black & white cow.

[101] Orr, G. (1999). Growing and pruning networks. https://www.willamette.edu/~gorr/classes/cs449/growing.html.

[102] Panda, P., Srinivasan, G., and Roy, K. (2017). Convolutional spike timing dependent plasticity based feature learning in spiking neural networks. *CoRR*, abs/1703.03854.

[103] Parkhi, O. M., Vedaldi, A., Zisserman, A., et al. (2015). Deep face recognition. In *BMVC*, volume 1, page 6.

[104] Paterson, L. (2013). US farmers using robots instead of migrants to milk cows.

[105] Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269.

[106] Peterson, L. E. (2009). K-nearest neighbor. *Scholarpedia*, 4(2):1883.

[107] Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical review letters*, 59(19):2229.

[108] Plunkett, K. and Elman, J. L. (1997). *Exercises in rethinking innateness: A handbook for connectionist simulations*. MIT Press.

[109] Pons, J., Lidy, T., and Serra, X. (2016). Experimenting with musically motivated convolutional neural networks. In *Content-Based Multimedia Indexing (CBMI), 2016 14th International Workshop on*, pages 1–6. IEEE.

[110] Pontil, M. and Verri, A. (1998). Support vector machines for 3d object recognition. *IEEE transactions on pattern analysis and machine intelligence*, 20(6):637–646.

[111] Prechelt, L. (1998). Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761 – 767.

[112] Rani, B., Nandhitha, N., and Manoharan, N. (2008). Euclidean distance based color image segmentation algorithm for dimensional characterization of lack of penetration from weld thermographs for on-line weld monitoring in gtaw. In *17th World Conference on Nondestructive Testing,, Shanghai, China*, pages 25–28.

[113] Reingold, E. (n.d.). Artificial neural networks technology.

[114] Riesenhuber, M. and Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025.

[115] Roberts, E. (n.d.). Neural networks - history.

[116] Roerdink, J. B. and Meijster, A. (2000). The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta informaticae*, 41(1, 2):187–228.

[117] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[118] Rowley, H., Baluja, S., Kanade, T., et al. (1998). Rotation invariant neural network-based face detection. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 38–44. IEEE.

[119] Russell, I. (n.d.). Brief history of neural networks.

[120] Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455.

[121] SAMSUNG ELECTRONICS (2017). Bixby vision. http://www.samsung.com/global/galaxy/apps/bixby/vision/.

[122] Shu, C., Ding, X., and Fang, C. (2011). Histogram of the oriented gradient for face recognition. *Tsinghua Science & Technology*, 16(2):216–224.

[123] Simard, P. Y., LeCun, Y. A., Denker, J. S., and Victorri, B. (1998). Transformation invariance in pattern recognition—tangent distance and tangent propagation. In *Neural networks: tricks of the trade*, pages 239–274. Springer.

[124] Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *2013 12th International Conference on Document Analysis and Recognition*, volume 2, pages 958–958. IEEE Computer Society.

[125] Sinha, U. (2017). Sift: Theory and practice. http://aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/.

[126] Sivertsen, A. H., Chu, C.-K., Wang, L.-C., Godtliebsen, F., Heia, K., and Nilsen, H. (2009). Ridge detection with application to automatic fish fillet inspection. *Journal of Food Engineering*, 90(3):317–324.

[127] Sohn, K. and Lee, H. (2012). Learning invariant representations with local transformations. *arXiv preprint arXiv:1206.6418*.

[128] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[129] Su, A., Sun, X., Zhang, Y., and Yu, Q. (2016). Efficient rotation-invariant histogram of oriented gradient descriptors for car detection in satellite images. *IET Computer Vision*, 10(7):634–640.

[130] Suard, F., Rakotomamonjy, A., Bensrhair, A., and Broggi, A. (2006). Pedestrian detection using infrared images and histograms of oriented gradients. In *Intelligent Vehicles Symposium, 2006 IEEE*, pages 206–212. IEEE.

[131] Szegedy, C., Toshev, A., and Erhan, D. (2013). Deep neural networks for object detection. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 2553–2561. Curran Associates, Inc.

[132] Tsai, G. (2010). Histogram of oriented gradients. *University of Michigan*.

[133] Vanrie, J., Béatse, E., Wagemans, J., Sunaert, S., and Van Hecke, P. (2002). Mental rotation versus invariant features in object perception from different viewpoints: an fmri study. *Neuropsychologia*, 40(7):917–930.

[134] Vreeken, J. (2003). Spiking neural networks, an introduction.

[135] Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066.

[136] Widrow, B. and Lehr, M. A. (1990). 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442.

[137] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

[138] Wiskott, L., Berkes, P., Franzius, M., Sprekeler, H., and Wilbert, N. (2011). Slow feature analysis. *Scholarpedia*, 6(4):5282.

[139] Wu, J., Cui, Z., Sheng, V. S., Zhao, P., Su, D., and Gong, S. (2013). A comparative study of sift and its variants. *Measurement science review*, 13(3):122–131.

[140] Wu, M. and Zhang, Z. (2010). Handwritten digit classification using the mnist data set. *Course project CSE802: Pattern Classification & Analysis*.

[141] Yang, X., Chen, Q., Zhou, S., and Wang, X. (2011). Deep belief networks for automatic music genre classification. In *INTERSPEECH*, pages 2433–2436. ISCA.

[142] Zhu, Q., Yeh, M.-C., Cheng, K.-T., and Avidan, S. (2006). Fast human detection using a cascade of histograms of oriented gradients. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1491–1498. IEEE.

[143] Zou, W., Zhu, S., Yu, K., and Ng, A. Y. (2012). Deep learning of invariant features via simulated fixations in video. In *Advances in Neural Information Processing Systems*, pages 3212–3220.

List of Figures

List of Tables

List of Equations

Nomenclature

Greek Symbols

α	Learning Rate

Other Symbols

f	Function
i,j,k	Neuron Index
o	Output
T	Target
t	Iteration Counter
w_{ij}	Weight between node i and node j

Acronyms / Abbreviations

ANN	Artificial Neural Network
BN	Batch Normalisation
BP	Back Propagation
CNN	Convolutional Neural Network
DNN	Deep Neural Network
FC	Fully Connected
HOG	Histogram of Orientated Gradients
KNN	K-Nearest Neighbour

$LSTM$ Long Short Term Memory

MLP Multi-Layer Perceptron

NN Neural Network

PCA Principle Component Analysis

RBM Restricted Boltzmann Machine

$SIFT$ Scale Invariant Feature Transform

Appendix A

Deep Dropout Artificial Neural Networks For Recognising Digits And Characters in Natural Images

This paper can be downloaded from Springer LNCS at : https://link.springer.com/chapter/10.1007/978-3-319-26561-2_4
A copy can also be found at : https://erikbarrow.com/dropout/

Appendix B

Selective Dropout for Deep Neural Networks

This paper can be downloaded from Springer LNCS at : https://link.springer.com/chapter/10.1007/978-3-319-46675-0_57
A copy can also be found at : https://erikbarrow.com/selective-dropout/

Appendix C

Ethics Approval

C.1 2014-2015

Low Risk Research Ethics Approval

Where NO human participants are involved and/or when using secondary data - Undergraduate or Postgraduate or Member of staff evaluating service level quality

Project Title

The use of Deep Learning in the Invariance problems associated with Object Recognition

Principal Investigator Certification

I believe that this project **does not require research ethics approval**.	X
I confirm that I have answered all relevant questions in the checklist honestly.	X
I confirm that I will carry out the project in the ways described in the checklist. I will immediately suspend research and request a new ethical approval if the project subsequently changes the information I have given in the checklist.	X

Principal Investigator

Name: Erik Barrow .....

Date: 08/01/2015.....

Student's Supervisor (if applicable)

I have read the checklist and confirm that it covers all the ethical issues raised by this project fully and frankly. I confirm that I have discussed this project with the student and agree that it does not require research ethics approval. I will continue to review ethical issues in the course of supervision.

Name: Chrisina Jayne .....

Date: 08/01/2015.....

Low Risk Research Ethics Approval Checklist

Applicant Details

Project Ref:	P31064
Full name:	Erik Barrow
Faculty:	[EC] Faculty of Engineering and Computing
Department:	[CM] Computing & The Digital Environment
Module Code:	
Supervisor:	Chrisina Jayne
Project title:	The use of Deep Learning in the Invariance problems associated with Object Recognition
Date(s):	01/10/2014
Created:	08/01/2015 10:54

Project Details

This project will look at current methods of object recognition with neural networks and current advancements in dealing with variance in images. The project will then look to build on and/or develop new methods to help with combating issues associated with variance in images.

Participants in your research

Questions	Yes	No
1. Will the project involve human participants?		X

Risk to Participants

Questions	Yes	No
2. Will the project involve human patients/clients, health professionals, and/or patient (client) data and/or health professional data?		X
3. Will any invasive physical procedure, including collecting tissue or other samples, be used in the research?		X
4. Is there a risk of physical discomfort to those taking part?		X
5. Is there a risk of psychological or emotional distress to those taking part?		X
6. Is there a risk of challenging the deeply held beliefs of those taking part?		X
7. Is there a risk that previous, current or proposed criminal or illegal acts will be revealed by those taking part?		X
8. Will the project involve giving any form of professional, medical or legal advice, either directly or indirectly to those taking part?		X

Risk to Researcher

Questions	Yes	No
9. Will this project put you or others at risk of physical harm, injury or death?		X
10. Will project put you or others at risk of abduction, physical, mental or sexual abuse?		X
11. Will this project involve participating in acts that may cause psychological or emotional distress to you or to others?		X
12. Will this project involve observing acts which may cause psychological or emotional distress to you or to others?		X
13. Will this project involve reading about, listening to or viewing materials that may cause psychological or emotional distress to you or to others?		X
14. Will this project involve you disclosing personal data to the participants other than your name and the University as your contact and e-mail address?		X
15. Will this project involve you in unsupervised private discussion with people who are not already known to you?		X
16. Will this project potentially place you in the situation where you may receive unwelcome media attention?		X
17. Could the topic or results of this project be seen as illegal or attract the attention of the security services or other agencies?		X
18. Could the topic or results of this project be viewed as controversial by anyone?		X

Informed Consent of the Participant

Questions	Yes	No
19. Are any of the participants under the age of 18?		X
20. Are any of the participants unable mentally or physically to give consent?		X
21. Do you intend to observe the activities of individuals or groups without their knowledge and/or informed consent from each participant (or from his or her parent or guardian)?		X

Participant Confidentiality and Data Protection

Questions	Yes	No
22. Will the project involve collecting data and information from human participants who will be identifiable in the final report?		X
23. Will information not already in the public domain about specific individuals or institutions be identifiable through data published or otherwise made available?		X
24. Do you intend to record, photograph or film individuals or groups without their knowledge or informed consent?		X
25. Do you intend to use the confidential information, knowledge or trade secrets gathered for any purpose other than this research project?		X

Gatekeeper Risk

Questions	Yes	No
26. Will this project involve collecting data outside University buildings?		X
27. Do you intend to collect data in shopping centres or other public places?		X
28. Do you intend to gather data within nurseries, schools or colleges?		X
29. Do you intend to gather data within National Health Service premises?		X

Other Ethical Issues

Questions	Yes	No
30. Is there any other risk or issue not covered above that may pose a risk to you or any of the participants?		X
31. Will any activity associated with this project put you or the participants at an ethical, moral or legal risk?		X

Other Documents submitted

REGISTRY RESEARCH UNIT

ETHICS REVIEW FEEDBACK FORM

(Review feedback should be completed within 10 working days)

Name of applicant: Erik Barrow .....

Faculty/School/Department: [Faculty of Engineering and Computing] Computing & The Digital Environment

Research project title: The use of Deep Learning in the Invariance problems associated with Object Recognition

Comments by the reviewer

1. Evaluation of the ethics of the proposal:

2. Evaluation of the participant information sheet and consent form:

3. Recommendation:

(Please indicate as appropriate and advise on any conditions. If there any conditions, the applicant will be required to resubmit his/her application and this will be sent to the same reviewer).

	Approved - no conditions attached
	Approved with minor conditions (no need to re-submit)
	Conditional upon the following – please use additional sheets if necessary (please re-submit application)
	Rejected for the following reason(s) – please use other side if necessary
X	Not required

Name of reviewer: Anonymous.....

Date: 08/01/2015.....

Project Title

The use of Deep Learning in the Invariance problems associated with Object Recognition

Comments

Comment	Posted
Low risk project	Chrisina Jayne 08/01/2015 11:19 AM

C.2 2015-2016

Certificate of Ethical Approval

Applicant:

Erik Barrow

Project Title:

The use of Deep Learning in the Invariance problems associated with Object Recognition

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:

11 September 2017

Project Reference Number:

P60985

Low Risk Research Ethics Approval

Project Title

The use of Deep Learning in the Invariance problems associated with Object Recognition

Record of Approval

Principal Investigator

I **request an ethics peer review** and confirm that I have answered all relevant questions in this checklist honestly.	X
I confirm that I will carry out the project in the ways described in this checklist. I will immediately suspend research and request new ethical approval if the project subsequently changes the information I have given in this checklist.	X
I confirm that I, and all members of my research team (if any), have read and agreed to abide by the Code of Research Ethics issued by the relevant national learned society.	X
I confirm that I, and all members of my research team (if any), have read and agreed to abide by the University's Research Ethics, Governance and Integrity Framework.	X

Name: Erik Barrow .....

Date: 17/08/2017 .....

Student's Supervisor (if applicable)

I have read this checklist and confirm that it covers all the ethical issues raised by this project fully and frankly. I also confirm that these issues have been discussed with the student and will continue to be reviewed in the course of supervision.

Name: Mark Eastwood .....

Date: 23/08/2017 .....

Reviewer (if applicable)

Date of approval by anonymous reviewer: 11/09/2017

Low Risk Research Ethics Approval Checklist

Project Information

Project Ref	P60985
Full name	Erik Barrow
Faculty	Faculty of Engineering, Environment and Computing
Department	Computing & The Digital Environment
Supervisor	Mark Eastwood
Module Code	ECCOM
EFAAF Number	
Project title	The use of Deep Learning in the Invariance problems associated with Object Recognition
Date(s)	01/10/2015 - 30/09/2017
Created	17/08/2017 11:27

Project Summary

This project will look at current methods of object recognition with neural networks and current advancements in dealing with variance in images. The project will then look to build on and/or develop new methods to help with combating issues associated with variance in images.

Names of Co-Investigators and their organisational affiliation (place of study/employer)	
Is the project self-funded?	NO
Who is funding the project?	
Has the funding been confirmed?	NO
Are you required to use a Professional Code of Ethical Practice appropriate to your discipline?	NO
Have you read the Code?	NO

Project Details

What is the purpose of the project?	The purpose of the project is to develop a new method for deep learning as part of artificial neural networks. This new method should attempt to solve the problem of object variance in image recognition by creating an invariant function.
What are the planned or desired outcomes?	The desired outcome is a method that can create invariantly recognise images with rotation variance introduced into the image. This will improve image recognition in applications that expect a large amount of rotation variance.
Explain your research design	The research is Quantitative. Experiments will be conducted on pre-made data sets widely used in the research field of image recognition. Results will be compared to a benchmark neural network that does not use the new method. This is a direct comparison between accuracy rates of the network.
Outline the principal methods you will use	Research is conducted using machine learning methods on readily available image datasets provided for research purposes. This type of research does not involve any questionnaires or human participants. Open source software widely used by researchers will be used to conduct experiments and typical machine learning conference papers and journals provide a source of background information for the project.
Are you proposing to use an external research instrument, validated scale or follow a published research method?	NO
If yes, please give details of what you are using	
Will your research involve consulting individuals who support, or literature, websites or similar material which advocates, any of the following: terrorism, armed struggles, or political, religious or other forms of activism considered illegal under UK law?	NO
Are you dealing with Secondary Data? (e.g. sourcing info from websites, historical documents)	YES
Are you dealing with Primary Data involving people? (e.g. interviews, questionnaires, observations)	NO
Are you dealing with personal or sensitive data?	NO
Is the project solely desk based? (e.g. involving no laboratory, workshop or off-campus work or other activities which pose significant risks to researchers or	YES

participants)	
Are there any other ethical issues or risks of harm raised by the study that have not been covered by previous questions?	NO
If yes, please give further details	

External Ethical Review

Question		Yes	No
1	Will this study be submitted for ethical review to an external organisation? (e.g. Another University, Social Care, National Health Service, Ministry of Defence, Police Service and Probation Office)		X
	If YES, name of external organisation		
2	Will this study be reviewed using the IRAS system?		X
3	Has this study previously been reviewed by an external organisation?		X

Risk of harm, potential harm and disclosure of harm

Question			Yes	No
1	Is there any significant risk that the study may lead to physical harm to participants or researchers?			X
	If YES, please explain how you will take steps to reduce or address those risks			
2	Is there any significant risk that the study may lead to psychological or emotional distress to participants?			X
	If YES, please explain how you will take steps to reduce or address those risks			
3	Is there any risk that the study may lead to psychological or emotional distress to researchers?			X
	If YES, please explain how you will take steps to reduce or address those risks			
4	Is there any risk that your study may lead or result in harm to the reputation of participants, researchers, or their employees, or any associated persons or organisations?			X
	If YES, please explain how you will take steps to reduce or address those risks			
5	Is there a risk that the study will lead to participants to disclose evidence of previous criminal offences, or their intention to commit criminal offences?			X
	If YES, please explain how you will take steps to reduce or address those risks			
6	Is there a risk that the study will lead participants to disclose evidence that children or vulnerable adults are being harmed, or at risk or harm?			X
	If YES, please explain how you will take steps to reduce or address those risks			
7	Is there a risk that the study will lead participants to disclose evidence of serious risk of other types of harm?			X
	If YES, please explain how you will take steps to reduce or address those risks			
8	Are you aware of the CU Disclosure protocol?		X	

Online and Internet Research

Question				Yes	No
1	Will any part of your study involve collecting data by means of electronic media (e.g. the Internet, e-mail, Facebook, Twitter, online forums, etc)?				X
	If YES, please explain how you will obtain permission to collect data by this means				
2	Is there a possibility that the study will encourage children under 18 to access inappropriate websites, or correspond with people who pose risk of harm?				X
	If YES, please explain further				
3	Will the study incur any other risks that arise specifically from the use of electronic media?				X
	If YES, please explain further				
4	Will you be using survey collection software (e.g. BoS, Filemaker)?				X
	If YES, please explain which software				
5	Have you taken necessary precautions for secure data management, in accordance with data protection and CU Policy?			X	
	If NO	please explain why not			
	If YES	Specify location where data will be stored	Experiments contain no personal data that requires data protection and only uses readily available public research datasets.		
		Planned disposal date	30/09/2018		
		If the research is funded by an external organisation, are there any requirements for storage and disposal?			X
		If YES, please specify details			

Project Title

The use of Deep Learning in the Invariance problems associated with Object Recognition

Comments

Comment	Posted
This is a very low risk project which uses only publicly available data, and no personal data is involved. All work is desk/computer-based.	Mark Eastwood 23/08/2017 09:10 PM

C.3 2016-2017

Certificate of Ethical Approval

Applicant:

Erik Barrow

Project Title:

The use of Deep Learning in the Invariance problems associated with Object Recognition

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:

11 September 2017

Project Reference Number:

P61162

Low Risk Research Ethics Approval

Project Title

The use of Deep Learning in the Invariance problems associated with Object Recognition

Record of Approval

Principal Investigator

I **request an ethics peer review** and confirm that I have answered all relevant questions in this checklist honestly.	X
I confirm that I will carry out the project in the ways described in this checklist. I will immediately suspend research and request new ethical approval if the project subsequently changes the information I have given in this checklist.	X
I confirm that I, and all members of my research team (if any), have read and agreed to abide by the Code of Research Ethics issued by the relevant national learned society.	X
I confirm that I, and all members of my research team (if any), have read and agreed to abide by the University's Research Ethics, Governance and Integrity Framework.	X

Name: Erik Barrow .....

Date: 04/09/2017 .....

Student's Supervisor (if applicable)

I have read this checklist and confirm that it covers all the ethical issues raised by this project fully and frankly. I also confirm that these issues have been discussed with the student and will continue to be reviewed in the course of supervision.

Name: Mark Eastwood .....

Date: 06/09/2017 .....

Reviewer (if applicable)

Date of approval by anonymous reviewer: 11/09/2017

Low Risk Research Ethics Approval Checklist

Project Information

Project Ref	P61162
Full name	Erik Barrow
Faculty	Faculty of Engineering, Environment and Computing
Department	Computing & The Digital Environment
Supervisor	Mark Eastwood
Module Code	ECCOM
EFAAF Number	
Project title	The use of Deep Learning in the Invariance problems associated with Object Recognition
Date(s)	01/10/2016 - 22/09/2017
Created	04/09/2017 11:37

Project Summary

This project will look at current methods of object recognition with neural networks and current advancements in dealing with variance in images. The project will then look to build on and/or develop new methods to help with combating issues associated with variance in images.

Names of Co-Investigators and their organisational affiliation (place of study/employer)	
Is the project self-funded?	NO
Who is funding the project?	
Has the funding been confirmed?	NO
Are you required to use a Professional Code of Ethical Practice appropriate to your discipline?	NO
Have you read the Code?	NO

Project Details

What is the purpose of the project?	The purpose of the project is to develop a new method for deep learning as part of artificial neural networks. This new method should attempt to solve the problem of object variance in image recognition by creating an invariant function.
What are the planned or desired outcomes?	The desired outcome is a method that can create invariantly recognise images with rotation variance introduced into the image. This will improve image recognition in applications that expect a large amount of rotation variance.
Explain your research design	The research is Quantitative. Experiments will be conducted on pre-made data sets widely used in the research field of image recognition. Results will be compared to a benchmark neural network that does not use the new method. This is a direct comparison between accuracy rates of the network.
Outline the principal methods you will use	Research is conducted using machine learning methods on readily available image datasets provided for research purposes. This type of research does not involve any questionnaires or human participants. Open source software widely used by researchers will be used to conduct experiments and typical machine learning conference papers and journals provide a source of background information for the project.
Are you proposing to use an external research instrument, validated scale or follow a published research method?	NO
If yes, please give details of what you are using	
Will your research involve consulting individuals who support, or literature, websites or similar material which advocates, any of the following: terrorism, armed struggles, or political, religious or other forms of activism considered illegal under UK law?	NO
Are you dealing with Secondary Data? (e.g. sourcing info from websites, historical documents)	YES
Are you dealing with Primary Data involving people? (e.g. interviews, questionnaires, observations)	NO
Are you dealing with personal or sensitive data?	NO
Is the project solely desk based? (e.g. involving no laboratory, workshop or off-campus work or other activities which pose significant risks to researchers or	YES

participants)	
Are there any other ethical issues or risks of harm raised by the study that have not been covered by previous questions?	NO
If yes, please give further details	

External Ethical Review

Question		Yes	No
1	Will this study be submitted for ethical review to an external organisation? (e.g. Another University, Social Care, National Health Service, Ministry of Defence, Police Service and Probation Office)		X
	If YES, name of external organisation		
2	Will this study be reviewed using the IRAS system?		X
3	Has this study previously been reviewed by an external organisation?		X

Risk of harm, potential harm and disclosure of harm

Question			Yes	No
1	Is there any significant risk that the study may lead to physical harm to participants or researchers?			X
	If YES, please explain how you will take steps to reduce or address those risks			
2	Is there any significant risk that the study may lead to psychological or emotional distress to participants?			X
	If YES, please explain how you will take steps to reduce or address those risks			
3	Is there any risk that the study may lead to psychological or emotional distress to researchers?			X
	If YES, please explain how you will take steps to reduce or address those risks			
4	Is there any risk that your study may lead or result in harm to the reputation of participants, researchers, or their employees, or any associated persons or organisations?			X
	If YES, please explain how you will take steps to reduce or address those risks			
5	Is there a risk that the study will lead to participants to disclose evidence of previous criminal offences, or their intention to commit criminal offences?			X
	If YES, please explain how you will take steps to reduce or address those risks			
6	Is there a risk that the study will lead participants to disclose evidence that children or vulnerable adults are being harmed, or at risk or harm?			X
	If YES, please explain how you will take steps to reduce or address those risks			
7	Is there a risk that the study will lead participants to disclose evidence of serious risk of other types of harm?			X
	If YES, please explain how you will take steps to reduce or address those risks			
8	Are you aware of the CU Disclosure protocol?		X	

Online and Internet Research

Question				Yes	No
1	Will any part of your study involve collecting data by means of electronic media (e.g. the Internet, e-mail, Facebook, Twitter, online forums, etc)?				X
	If YES, please explain how you will obtain permission to collect data by this means				
2	Is there a possibility that the study will encourage children under 18 to access inappropriate websites, or correspond with people who pose risk of harm?				X
	If YES, please explain further				
3	Will the study incur any other risks that arise specifically from the use of electronic media?				X
	If YES, please explain further				
4	Will you be using survey collection software (e.g. BoS, Filemaker)?				X
	If YES, please explain which software				
5	Have you taken necessary precautions for secure data management, in accordance with data protection and CU Policy?			X	
	If NO	please explain why not			
	If YES	Specify location where data will be stored	Experiments contain no personal data that requires data protection and only uses readily available public research datasets.		
		Planned disposal date	30/09/2018		
		If the research is funded by an external organisation, are there any requirements for storage and disposal?			X
		If YES, please specify details			

Project Title

The use of Deep Learning in the Invariance problems associated with Object Recognition

Comments

Comment	Posted
Low risk; no change from previous year's approval	Mark Eastwood 06/09/2017 06:01 PM

Appendix D

Library Declaration and Deposit Form